AERODYNAMIC DESIGN APPLYING AUTOMATIC DIFFERENTIATION AND USING ROBUST VARIABLE FIDELITY OPTIMIZATION

A Thesis
Presented to
The Academic Faculty

by

Takemiya Tetsushi

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
December 2008

AERODYNAMIC DESIGN APPLYING AUTOMATIC DIFFERENTIATION AND USING ROBUST VARIABLE FIDELITY OPTIMIZATION

Approved by:

Professor Dimitri Mavris,
Committee Chair
School of Aerospace Engineering
Georgia Institute of Technology

Professor Lakshmi Sankar
School of Aerospace Engineering
Georgia Institute of Technology

Professor Stephen Ruffin
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Alley Nicholas
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Sriram Rallabhandi

National Institute of Aerospace

Date Approved: August 22, 2008

To my parents

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr. Dimitri Mavris for giving me an opportunity to come and work in the Aerospace System Design Laboratory at Georgia Institute of Technology. He was so friendly to me, and not only his deep technical expertise but also his Japanese greetings always cheered me up when I had difficulties in my research. I also would like to thank my committee members, Dr. Sankar, Dr. Ruffin, Dr. Alley, and Dr. Rallabhandi. Especially, Dr. Ruffin always answered me precisely when I had questions about aerodynamics, and Dr. Rallabhandi always advised me when I faced difficulties in my research.

I would like to thank Dr. Fujii, my former advisor in the University of Tokyo. He was willing to give me permission for using his CFD code LANS in my doctoral thesis. My work would not be complete without his kindness.

I met Dr. Kamio, a biochemistry scientist at Georgia State University. Although our research fields are different, he advised me about research as well as personal issues. I really appreciate him.

Ms. Jane Chisholm and Mr. Curtis Iwata helped me to write the thesis in English, and Dr. Miyabe Shigeki and Mr. Matsuoka Yamato helped me with editing in LaTeX. Thank you.

I have many friends in Atlanta. Justin Ducote, one of my best friends, introduced me many of his (now our) friends and showed me the American culture in our generation. Thanks to Justin, my life in Atlanta has become very exciting. Thank you Justin!

I also appreciate my friends in Japan. Ryudo, Noriyo, Atushi, Hiroo,... They always gave me energy. Although they live very far from the US, they were always

very close to me in my mind.

Finally, I would like to thank my parents, who have been supporting me throughout my entire life with their love. My mother always cheered me up with her positive personality, and my father showed me his braveness in his life. This thesis would not have been possible with them. Thank you so much.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS OR ABBREVIATIONS

AD	Automatic Differentiation
AMF	Approximate Management Framework
β	Shape parameter
C_d	Drag coefficient (2D)
C_D	Drag coefficient (3D)
CFD	Computational Fluid Dynamics
C_l	Lift coefficient (2D)
C_L	Lift coefficient 3D)
C_m	Moment coefficient (2D)
C_p	Pressure coefficient
$\mathbf{\Delta}_n$	Trust region size at iteration n
FAD	Forward Automatic Differentiation
FD	Finite Differentiation
f_{high}	Objective function value evaluated by a high-fidelity model
f_{low}	Objective function value evaluated by a low-fidelity model
FPE	Full Potential Equation
$f_{surrogate}$	Objective function value evaluated by a surrogate model
g_{high}	Constraint value evaluated by a high-fidelity model
g_{low}	Constraint value evaluated by a low-fidelity model
$g_{surrogate}$	Constraint function value evaluated by a surrogate model
$\mathbf{H}_n$	Quasi Hessian Matrix at iteration n
PDE	Partial Differential Equation
ϕ	Penalty function
$\mathbf{Q}_0$	Initial state vector
$\mathbf{Q}_{converged}$	Converged state vector

$\mathbf{Q}_n$	State vector at iteration n
RAD	Reverse Automatic Differentiation
ρ_n	Trust region ratio at iteration n
r_p	Penalty weight
S	Enclosed airfoil area
SQP	Sequential Quadratic Programming
tolg	Tolerance for constraints
VFO	Variable Fidelity optimization
$\mathbf{x}$	Grid points
$\mathbf{x}_0$	Initial design point
$\mathbf{x}_c$	Candidate design point
$\mathbf{x}_n$	Design point at iteration n

SUMMARY

In modern aerospace engineering, the physics-based computational design method is becoming more important, as it is more efficient than experiments and because it is more suitable in designing new types of aircraft (e.g., unmanned aerial vehicles or supersonic business jets) than the conventional design method, which heavily relies on historical data. To enhance the reliability of the physics-based computational design method, researchers have made tremendous efforts to improve the fidelity of models. However, high-fidelity models require longer computational time, so the advantage of efficiency is partially lost. This problem has been overcome with the development of variable fidelity optimization (VFO). In VFO, different fidelity models are simultaneously employed in order to improve the speed and the accuracy of convergence in an optimization process.

Among the various types of VFO methods, one of the most promising methods is the approximation management framework (AMF). In the AMF, objective and constraint functions of a low-fidelity model are scaled at a design point so that the scaled functions, which are referred to as "surrogate functions," match those of a high-fidelity model. Since scaling functions and the low-fidelity model constitutes surrogate functions, evaluating the surrogate functions is faster than evaluating the high-fidelity model. Therefore, in the optimization process, in which gradient-based optimization is implemented and thus many function calls are required, the surrogate functions are used instead of the high-fidelity model to obtain a new design point. The best feature of the AMF is that it may converge to a local optimum of the high-fidelity model in much less computational time than the high-fidelity model.

However, through literature surveys and implementations of the AMF, the author

found that 1) the AMF is very vulnerable when the computational analysis models have numerical noise, which is very common in high-fidelity models, and that 2) the AMF terminates optimization erroneously when the optimization problems have constraints. The first problem is due to inaccuracy in computing derivatives in the AMF, and the second problem is due to erroneous treatment of the trust region ratio, which sets the size of the domain for an optimization in the AMF.

In order to solve the first problem of the AMF, automatic differentiation (AD) technique, which reads the codes of analysis models and automatically generates new derivative codes based on some mathematical rules, is applied. If derivatives are computed with the generated derivative code, they are analytical, and the required computational time is independent of the number of design variables, which is very advantageous for realistic aerospace engineering problems. However, if analysis models implement iterative computations such as computational fluid dynamics (CFD), which solves system partial differential equations iteratively, computing derivatives through the AD requires a massive memory size. The author solved this deficiency by modifying the AD approach and developing a more efficient implementation with CFD, and successfully applied the AD to general CFD software.

In order to solve the second problem of the AMF, the governing equation of the trust region ratio, which is very strict against the violation of constraints, is modified so that it can accept the violation of constraints within some tolerance. By accepting violations of constraints during the optimization process, the AMF can continue optimization without terminating immaturely and eventually find the true optimum design point.

With these modifications, the AMF is referred to as "Robust AMF," and it is applied to airfoil and wing aerodynamic design problems using Euler CFD software. The former problem has 21 design variables, and the latter 64. In both problems, derivatives computed with the proposed AD method are first compared with those

computed with the finite differentiation (FD) method, and then, the Robust AMF is implemented along with the sequential quadratic programming (SQP) optimization method with only high-fidelity models. The proposed AD method computes derivatives more accurately and faster than the FD method, and the Robust AMF successfully optimizes shapes of the airfoil and the wing in a much shorter time than SQP with only high-fidelity models. These results clearly show the effectiveness of the Robust AMF.

Finally, the feasibility of reducing computational time for calculating derivatives and the necessity of AMF with an optimum design point always in the feasible region are discussed as future work.

CHAPTER I

INTRODUCTION

Modern aerospace engineers have applied their skills and ingenuity to developing and designing myriad types of aircraft, including supersonic business jets (SBJ), unmanned aerial vehicles (UAV), and the Mars airplane (Figure 1). Since these aircraft have had a short history, conventional design methods [66][10][70] that heavily rely on historical data are not suitable for their design. Thus, the design of such revolutionary aircraft calls for physics-based design methods, which have received increasing attention in modern aerospace engineering.

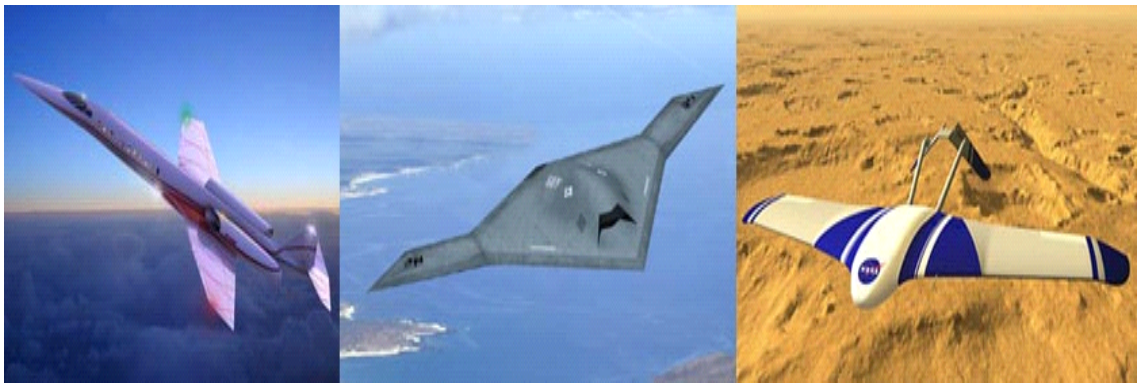


Figure 1: New types of aircraft.

From the perspective of integrated product and process development (IPPD) [4], using better analysis methods in the earlier phases of design allows engineers to explore a broader scope of design features, leading to a better design with reduced

production costs and production time. This concept is depicted in Figure 2 [57], in which dashed lines represent a traditional design process that often relies on non-physical data and historical regression and the solid lines represent an IPPD design process that relies on better analysis methods. Generally, the better analysis methods are physics based, particularly when the aircraft is state of the art. Therefore, from the IPPD point of view as well, one can conclude that using physics-based design methods is preferable.

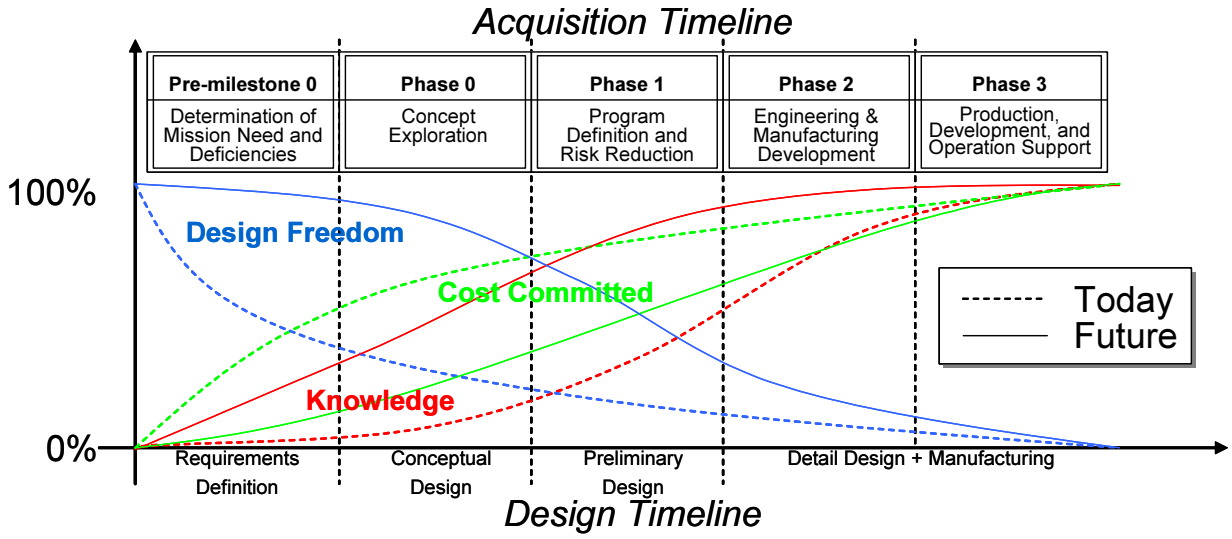


Figure 2: Different stages of design [57].

Here, one may question the physics-based analysis: What kind of physics-based analysis tools are either available or under development? How long do they take to implement? Are they suitable in a design process in which many iterations are required to obtain an optimum solution? These questions are discussed in the context of aerodynamics in the following section.

1.1 Physics-Based Analysis in Aerodynamics

Figure 3 shows different flow-governing equations, ordered from the most to the least accurate. Consequently, those higher on the list take more time to implement. The time-scale bar shown on the left-hand side of Figure 3 is relatively rough, but time orders of different flow-governing equations are correct if they are used in the same condition.

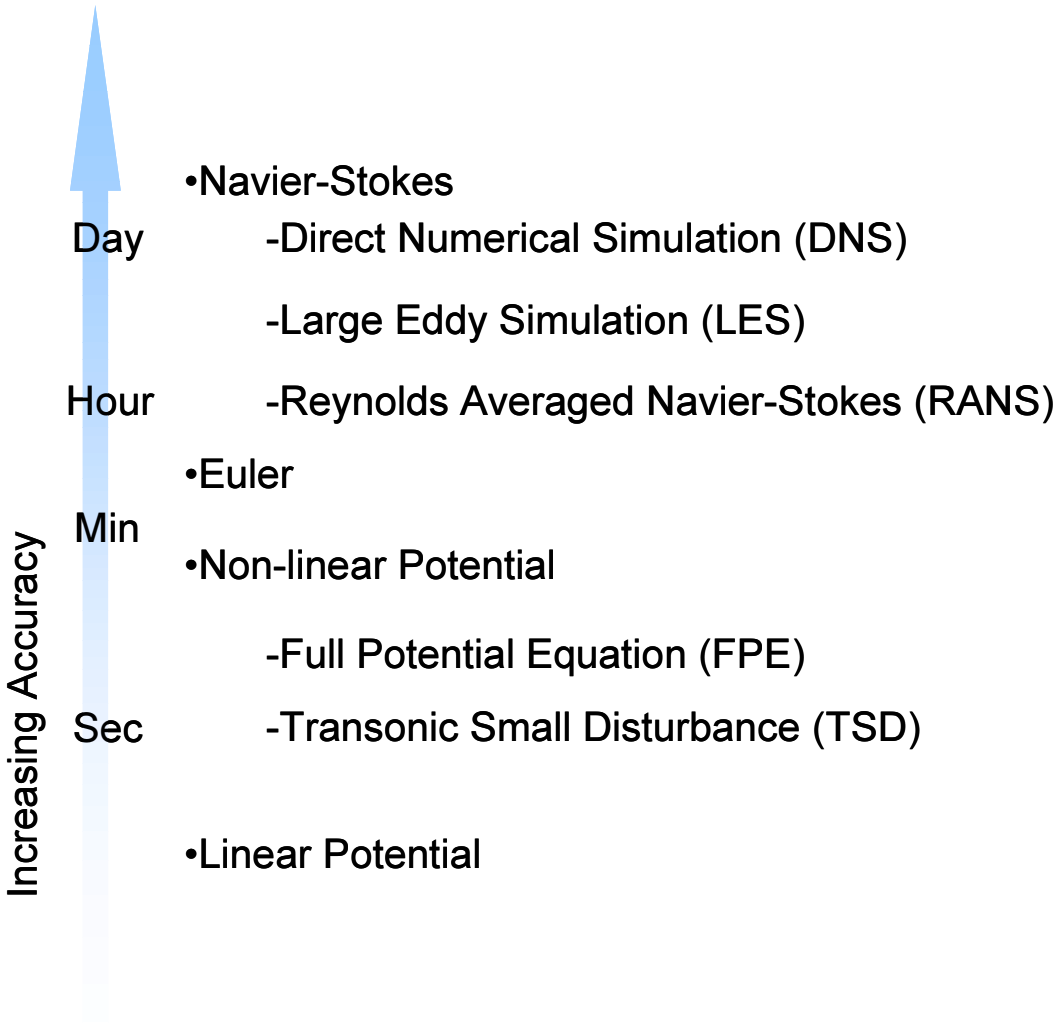


Figure 3: Different-fidelity, flow-governing equations.

The governing equations in Figure 3 are briefly explained below.

Direct numerical simulation (DNS)

DNS solves discretized Navier-Stokes equations numerically without any turbulence model. That is, the entire range of spatial and temporal scales of turbulence must be resolved. Therefore, DNS requires many small cells, and mathematically, the number of cells required is known to be on the order of $R_e{}^{\frac{9}{4}}$ [86], where R_e represents the Reynolds number of the analyzed flow. Currently using DNS for aerospace engineering problems in which R_e is generally over 10^6 is impractical due to computational limits. Thus, DNS is used for relatively simple flows at low Reynolds numbers and for the validation of turbulence models [77].

Large eddy simulation (LES)

LES solves discretized Navier-Stokes equations numerically with a spatially-averaged turbulence model. Here, while a large eddy is directly resolved, a small eddy is resolved with only the turbulence model. Since LES is able to predict instantaneous flow characteristics and resolve turbulent flow structures, it is suitable in simulations involving chemical reactions such as fuel combustion of in engines. The computational effort required for LES is less than that of DNS by a factor of approximately ten [77]. However, due to computational limits, using LES in optimization problems is not yet practical.

Reynolds-averaged Navier-Stokes (RANS)

RANS solves discretized Reynolds-averaged Navier-Stokes equations numerically with a temporally-averaged turbulence model. In order to capture the profile of the boundary layer, RANS still requires small cells near the wall ($\Delta y_{min} \doteq \frac{0.01}{\sqrt{Re}}$ [35]), but as it uses the turbulence model for any type of eddy, the computational cost is lower

than it is for DNS and LES. Therefore, RANS is widely used in practical aerospace engineering problems. However, in optimization problems, for which many iterations are required, RANS still requires a very powerful computational environment (i.e., a super computer, cluster computers).

Euler

Euler neglects viscous terms in RANS, so it can be used for problems in which the boundary layer does not play an important role. Since Euler requires a lower number of cells than other higher-fidelity governing equations due to the neglect of viscous terms, it is computationally cheaper and thus widely used in optimization problems.

Non-linear potential

A Non-linear potential equation is also referred to as a "velocity potential equation" since it is described by a velocity potential. Depending on the degree of assumptions, a non-linear potential equation can be either a full potential equation (FPE) or a transonic small disturbance (TSD) [9]. The FPE is obtained if irrotational and isentropic conditions are added to the Euler equation. Since a shock wave involves a change of entropy, a non-linear potential can be used only when a shock wave is relatively weak. Furthermore, if the assumption that the flow is only slightly perturbed from uniform free stream conditions is added to the FPE, the TSD is obtained. This assumption is justified when the analyzed geometry has a slender body, when it has a small camber, or when it is flying with a small angle of attack. Jameson [47] created a method referred to as a "rotated difference scheme" for solving the FPE, and Murman and Cole [59] created a method for solving the TSD.

Linear potential

If the TSD is linearized, a linear potential equation can be obtained. In a linear

potential equation, a shock wave cannot be captured since it is a phenomenon of non-linearity. Therefore, a linear potential equation should be used in either a subsonic or supersonic speed regime. If a Prandtl-Glauert transformation [11] is implemented, the linear potential equation becomes a Laplace equation, which can be discretized and solved numerically; however, since the basic element solutions of a Laplace equation are mathematically known (e.g., source, sink, doublet, and vortex), and since it is linear, superimposing these element solutions in order to compute aerodynamics is possible if appropriate boundary conditions are given. This characteristic is unique, so many methods for computing aerodynamics governed by the linear potential exist. The panel method [15], for example, distributes the basic element solutions on the panels of objects (i.e., a discretized surface) and computes the strengths of the basic solutions so that they satisfy the given boundary conditions on the control points on the panels (Figure 4). Under the assumption that the stream line is parallel to the panels, the vortex lattice method [16] flattens an object onto a sheet, divides it into many panels, distributes horseshoe vortices, and computes their strengths (Figure 5).

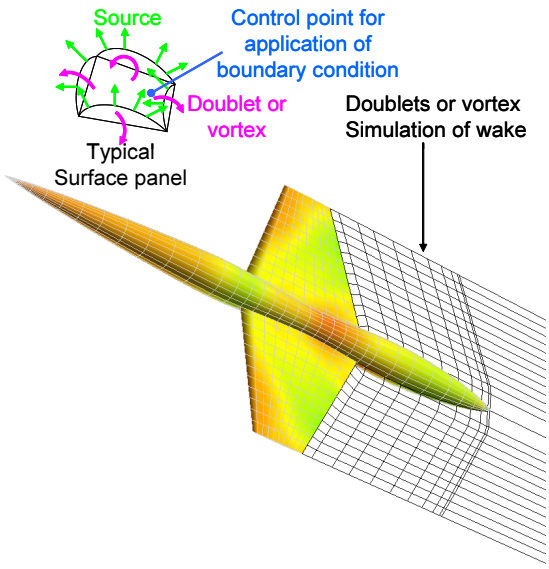


Figure 4: The panel method implemented by PANAIR [71].

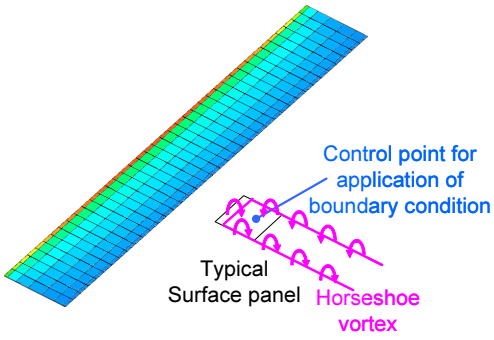


Figure 5: The vortex lattice method.

1.2 Computing Higher-Fidelity Models More Rapidly

As discussed in the previous section, higher-fidelity flow governing equations are more physics based, but some are not yet suitable for design features that require many iterations because their implementation requires huge memory and considerable time. On the other hand, using higher-fidelity models up front is very important, as was shown in Figure 2. In order to resolve this dilemma, two solutions may exist: to develop better hardware or to develop better software. The following subsections discuss each approach in more detail.

1.2.1 Developing Better Hardware

Computer simulations partially depend on computational power, which has been continuously increasing. Indeed, simulations that once had to be run on a super computer can now be run on a personal computer (PC) within a reasonable time period today. Therefore, reviewing the history of computers and discussing their future are important to dissertation.

Figure 6, released by an organization referred to as "Top500 [3]," shows the history and the prediction of floating point number operations per second (FLOPS) in

supercomputers. FLOPS are directly related to computational performance in scientific calculations. For example, a supercomputer with 10G FLOPS can conduct 10^{10} calculations per second. From Figure 6, one can conclude that the speed of computations in supercomputers will be roughly 1,000 times greater in ten years.

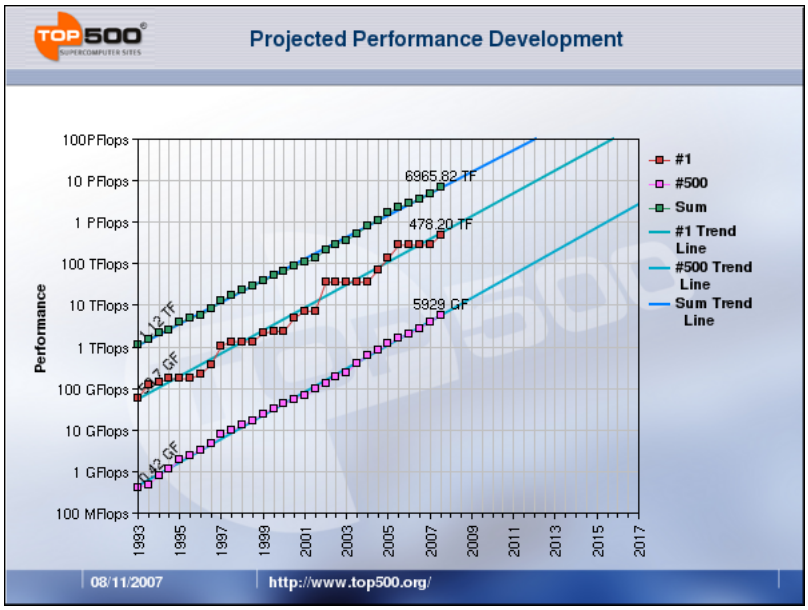


Figure 6: The prediction of FLOPS in supercomputers [3].

These days, PCs, widely used for all purposes, are becoming more popular in scientific computations as well. However, measuring computational performance in PCs is slightly complicated because FLOPS are not commonly used; instead, a clock number is widely used. Here, a larger clock number does not necessarily indicate faster scientific computations. However, benchmark tests have shown that recent PCs contain several hundred mega FLOPS [28], which are equivalent to the number in supercomputers in the early to mid-1990's. In addition, Ekman [25] reported that the performance growth of PCs between 1996 to 2004 was 41%, a trend expected to continue for at least several years.

1.2.2 Developing Better Software

Another choice for speeding up optimizations is to modify analytical codes. A good example of modifying analytical software without degrading their fidelity is to rewrite analytical codes so that they can be used in parallel computing or replace explicit schemes with implicit schemes in CFD codes. Another choice is to develop better optimization frameworks. For instance, using sequential quadratic programming (SQP) [80] is usually faster and better in convergence than linear programming (LP) [81]. Another interesting method is to couple different fidelity models mathematically, referred to as "variable fidelity optimization" (VFO). In VFO, the models do not have to be modified, but one can expect to obtain a better solution with only a lower-fidelity model and implement it in a shorter time period than one can with a higher-fidelity model.

1.3 Motivation

As mentioned in the previous section, several approaches can speed up optimizations with higher-fidelity models or quasi-higher-fidelity models, depicted notionally in Figure 7.

With regard to hardware, several potential approaches for increasing simulation speed, such as modifying the CPU, are available, but outside of the scope of this research. With regard to software, however, several hierarchical approaches are available since analytical codes work inside optimization frameworks, represented in Figure 7. In other words, modifications of the optimization framework impact the speed of optimizations regardless of which analytical codes are used. In addition, a VFO-type framework allows one to interchange older or lower-fidelity codes with newer or higher-fidelity codes, depending on the application. This option is very attractive

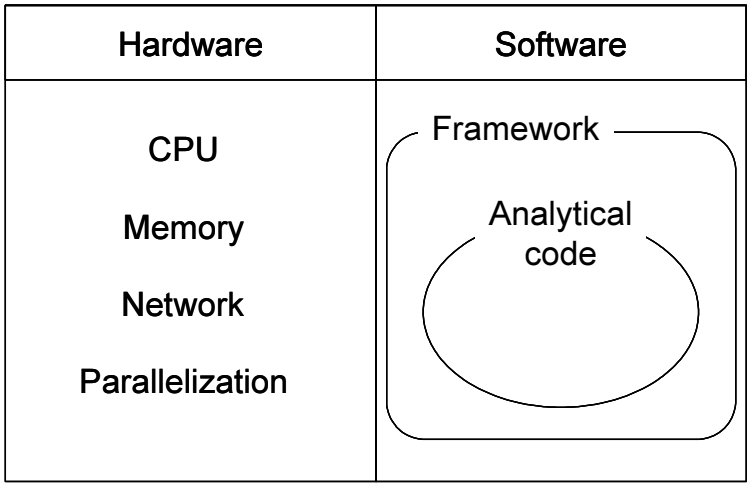


Figure 7: Approaches in speeding up optimizations with higher-fidelity models or quasi-higher-fidelity models.

when multiple analysis codes are available. Therefore, this dissertation starts by surveying modern VFO-type optimization frameworks and then focuses on developing a better VFO-type optimization framework.

1.4 Dissertation Overview

The remainder of this dissertation is organized as shown in Figure 8. Chapter II surveys existing VFO methods and categorizes them so that one can identify which method is preferable in aerospace engineering problems. After the selection of a VFO method, Chapter III focuses on the approximate management framework (AMF), one of the VFO methods surveyed in Chapter II, and then explains the AMF in detail and tests it using example optimization problems that reveal the technical barriers and flaws of the AMF. Chapter IV presents hypotheses for solving these research questions and establishes a new framework referred to as the "Robust AMF." Chapter V implements the Robust AMF on the optimization of an airfoil design and a wing design, and Chapter VI concludes the dissertation and describes future work based on the implementation of the Robust AMF.

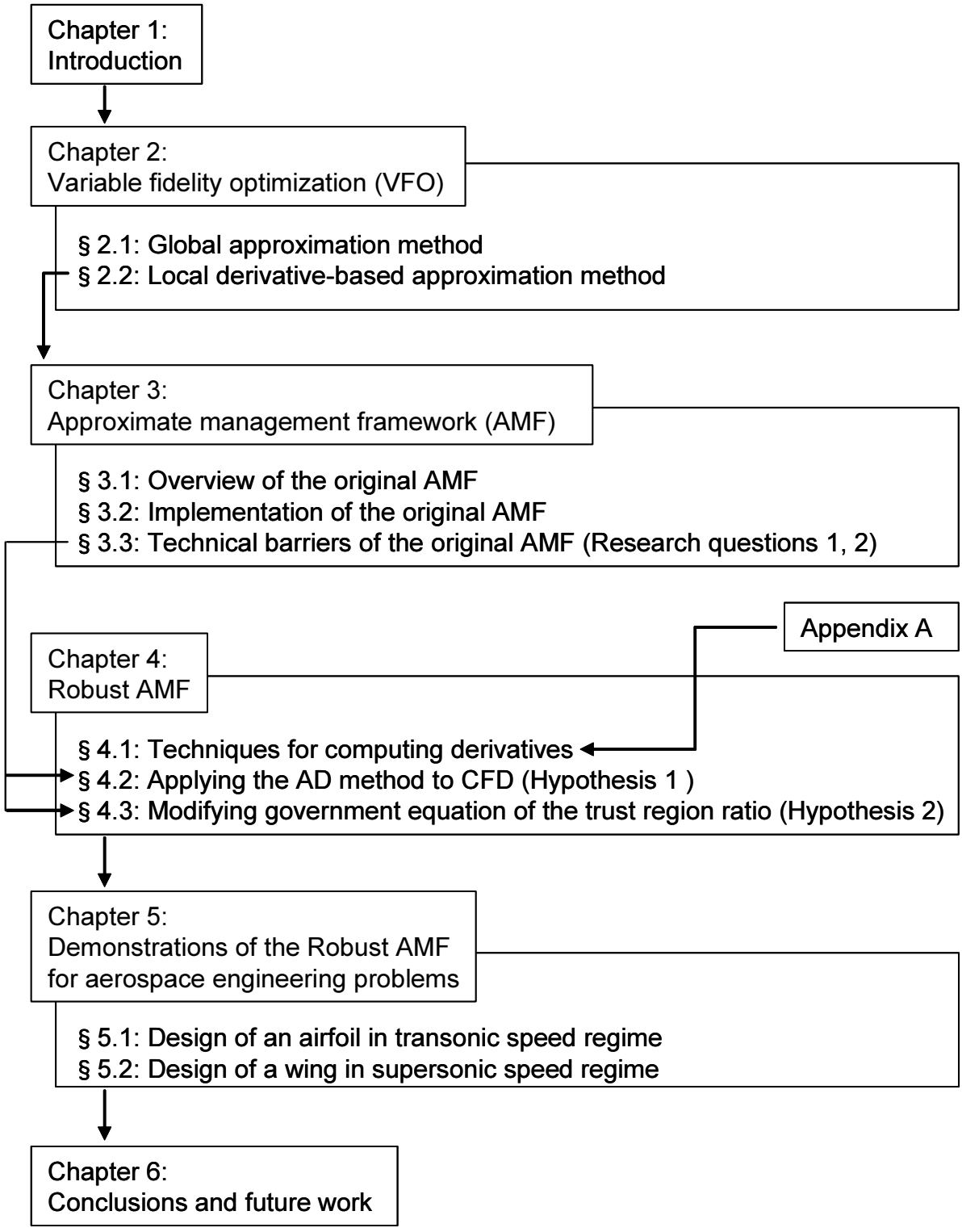


Figure 8: Dissertation overview.

CHAPTER II

VARIABLE FIDELITY OPTIMIZATION (VFO)

VFO uses high-fidelity and low-fidelity models simultaneously to reduce optimization time while converging on a solution for the high-fidelity model. Depending on the way surrogate models of high-fidelity models are created, VFO can be classified into two groups: One uses global approximation to capture the behavior of the objective function and constraints over the entire design domain; and the second uses local derivative-based approximation such as polynomial approximation, based on a Taylor-series expansion about a design point. In the following sections, these two groups are explained in greater detail and summarized in Table 1 at the end of this chapter.

2.1 Global Approximation Method

The simplest and probably the most widely used global approximation method is the second-order polynomial response surface method (RSM) [32][51]. However, since second-order polynomial equations cannot capture non-linear effects accurately, other meta-modeling such as the Neural network [41], Kriging [72], cokriging [20] and Gaussian processes (GP) [65] are becoming more populer, as they can capture non-linear effects. One important characteristic about Kriging, cokriging, and GP is that they can include the trends of the functions of interest if they are known beforehand [61]. If the trends of the functions are given, the number of samples required to construct a surrogate model can be reduced dramatically. Using this characteristic, El-Beltagy [26] developed a VFO framework referred to as the "fusion framework."

Generally, global approximation is less elegant than local derivative-based approximation, but it may produce an acceptable level of accuracy over the entire design

domain. Furthermore, global approximation requires a smart sampling method such as the design of experiments (DOE) [33] to minimize the number of function evaluations. However, even with the sophisticated DOE, sampling becomes more difficult as the number of design variables increases and as functions become more non-linear. In addition, the number of design variables in the global approximation method is generally limited to about fifteen.

Once a global approximation (i.e., a low-fidelity model) is constructed, an optimum design point is found with an optimizer. By iteratively reconstructing the global model by adding a newly found optimum design point, one can develop a better surrogate model and obtain a better optimum design point. This optimization framework is referred to as the "successive approximate optimization" (SAO) algorithm [39].

2.2 Local Derivative-Based Approximation Method

The idea of the local derivative-based approximation originated in a study by Chang [19] in 1993. Its methodology can be divided into two categories, depending on whether the surrogate models are scaled using multiplicative scaling or additive scaling.

2.2.1 Multiplicative Scaling

Chang analyzed the structural responses of aircraft components, such as wing tip displacement, stress, and frequencies, by coupling different fidelity models with the multiplicative approach, as shown in Equations 1 and 2, where the low-fidelity model is multiplied by a scaling function to approximate the high-fidelity model, and the scaling function is expressed by a Taylor series expanded at $\mathbf{x}_n$.

$$f_{high}(\mathbf{x}) = \frac{f_{high}(\mathbf{x})}{f_{low}(\mathbf{x})} \times f_{low}(\mathbf{x}) \equiv \beta(\mathbf{x}) \times f_{low}(\mathbf{x}), \quad (1)$$

$$\beta(\mathbf{x}) = \beta(\mathbf{x}_n) + \nabla\beta(\mathbf{x}_n)^T(\mathbf{x}-\mathbf{x}_n) + \frac{1}{2}(\mathbf{x}-\mathbf{x}_n)^T\nabla^2\beta(\mathbf{x}_n)(\mathbf{x}-\mathbf{x}_n) + \cdots \quad (2)$$

As a reference, the zeroth, first, and second differentials of $\beta(\mathbf{x})$ at $\mathbf{x}_n$ are shown below:

$$\beta(\mathbf{x}_n) = \frac{f_{high}(\mathbf{x}_n)}{f_{low}(\mathbf{x}_n)}. \quad (3)$$

$$\nabla\beta(\mathbf{x}_n) = \frac{1}{f_{low}(\mathbf{x}_n)}\nabla f_{high}(\mathbf{x}_n) - \frac{f_{high}(\mathbf{x}_n)}{[f_{low}(\mathbf{x}_n)]^2}\nabla f_{low}(\mathbf{x}_n). \quad (4)$$

$$\begin{aligned}\nabla^2\beta(\mathbf{x}_n) &= \frac{1}{f_{low}(\mathbf{x}_n)}\nabla^2 f_{high}(\mathbf{x}_n) - \frac{f_{high}(\mathbf{x}_n)}{[f_{low}(\mathbf{x}_n)]^2}\nabla^2 f_{low}(\mathbf{x}_n) \\ &+ \frac{2f_{high}(\mathbf{x}_n)}{[f_{low}(\mathbf{x}_n)]^3}\nabla f_{low}(\mathbf{x}_n)\nabla f_{low}^T(\mathbf{x}_n) \\ &- \frac{1}{[f_{low}(\mathbf{x}_n)]^2}\left[\nabla f_{low}(\mathbf{x}_n)\nabla f_{high}^T(\mathbf{x}_n) + \nabla f_{high}(\mathbf{x}_n)\nabla f_{low}^T(\mathbf{x}_n)\right]. \quad (5)\end{aligned}$$

However, as Marduel [54] and Gano [36] pointed out, when $f_{low}(\mathbf{x})$ is close to zero, the multiplicative approach encounters a problem referred to as the "zero-divided problem" because it appears in the denominator in Equation 1.

2.2.2 Additive Scaling

In order to overcome the zero-divided problem in multiplicative scaling, Giunta [37] introduced the additive approach, as shown in Equations 6 and 7. Giunta [37] and Marduel [54] reported that the additive scaling method is preferable in almost all cases and poses no mathematical difficulties.

$$f_{high}(\mathbf{x}) = \gamma(\mathbf{x}) + f_{low}(\mathbf{x}), \quad (6)$$

$$\gamma(\mathbf{x}) = \gamma(\mathbf{x}_n) + \nabla\gamma(\mathbf{x}_n)^T(\mathbf{x}-\mathbf{x}_n) + \frac{1}{2}(\mathbf{x}-\mathbf{x}_n)^T\nabla^2\gamma(\mathbf{x}_n)(\mathbf{x}-\mathbf{x}_n) + \cdots \quad (7)$$

As a reference, the zeroth, first, and second differentials of $\gamma(\mathbf{x})$ at $\mathbf{x}_n$ are shown below:

$$\gamma(\mathbf{x}_n) = f_{high}(\mathbf{x}_n) - f_{low}(\mathbf{x}_n). \tag{8}$$

$$\nabla\gamma(\mathbf{x}_n) = \nabla f_{high}(\mathbf{x}_n) - \nabla f_{low}(\mathbf{x}_n). \tag{9}$$

$$\nabla^2\gamma(\mathbf{x}_n) = \nabla^2 f_{high}(\mathbf{x}_n) - \nabla^2 f_{low}(\mathbf{x}_n). \tag{10}$$

Although the local derivative-based approximation method is generally more efficient than the global approximation method, the former may end up finding a local minimum. In addition, it demands the accurate and efficient computation of derivatives which may be obtained by using either the adjoint method [48] or the automatic differentiation (AD) method [84]. One important characteristic of the local derivative-based approximation method is that it can deal with many design variables. Indeed, Hutchison [45] and Dudley [24] used more than 50 design variables to optimize a wing, but their analysis tools are rather simple.

Once the local derivative-based approximation is constructed, an optimum design point is found with an optimizer. By iteratively reconstructing the local derivative-based model systematically, one can implement the approximation management framework (AMF) created by Alexandrov [6] to obtain a better optimum design point.

2.2.3 Approximating Second-Order Information at a Low Cost

In local derivative-based approximation methods, using higher-order scaling functions are better because they lead to more accurate surrogate functions; but they also require more computational time because they require higher-order derivatives. By trading off accuracy and computational time, first- or second-order scaling is probably the most suitable for practical problems. When using second-order scaling, the Hessian matrix must be obtained. However, the Hessian matrix, which contains the

second-order partial derivative information of the function, is very costly to compute. Fortunately, when the surrogate models are iteratively updated such as the AMF, mathematical techniques can approximate the Hessian matrix from the first-order information. In this case, the approximated Hessian matrix approaches the true Hessian matrix asymptotically as the number of iterations increases. Since the scaling function with this quasi-Hessian matrix is more accurate than that with the first-order information and since it is less costly than that with the true Hessian matrix, the quasi-Hessian matrix is probably a better option for constructing the surrogate functions.

The most prevalent method for computing the quasi-Hessian matrix is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update [18], [31], [38], [75]. In BFGS, the initial quasi Hessian is the identity matrix ($\mathbf{H}_1 = \mathbf{I}$), and the following scaled-identity matrix is used as the second quasi Hessian:

$$\mathbf{H}_2 = \frac{\mathbf{y}_2^T \mathbf{s}_2}{\mathbf{y}_2^T \mathbf{y}_2} \mathbf{I}, \tag{11}$$

where

$$\mathbf{y}_2 = \nabla f_2 - \nabla f_1, \tag{12}$$

$$\mathbf{s}_2 = \mathbf{x}_2 - \mathbf{x}_1. \tag{13}$$

In Equation 12, f is the function for which the quasi-Hessian information is desired. Then, the following matrix is used as the quasi Hessian after the third iteration:

$$\mathbf{H}_{n+1} = \begin{cases} \mathbf{H}_n - \frac{\mathbf{H}_n \mathbf{s}_n \mathbf{s}_n^T \mathbf{H}_n}{\mathbf{s}_n^T \mathbf{H}_n \mathbf{s}_n} + \frac{\mathbf{y}_n \mathbf{y}_n^T}{\mathbf{y}_n^T \mathbf{s}_n} & \text{if } \left|\mathbf{y}_n^T \mathbf{s}_n\right| \geq 10^{-6} \mathbf{s}_n^T \mathbf{H}_n \mathbf{s}_n \\ & n \geq 2, \\ \mathbf{H}_n & \text{otherwise} \end{cases} \tag{14}$$

where

$$\mathbf{y}_{n+1} = \nabla f_{n+1} - \nabla f_n, \tag{15}$$

$$\mathbf{s}_{n+1} = \mathbf{x}_{n+1} - \mathbf{x}_n. \tag{16}$$

2.3 Summary of the VFO

A summary of the VFO is shown in Table 1.

Table 1: Classification of the VFO.

	Global Approximation Method	Local Derivative-Based Approximation Method
Merits	• Creates global models	• Creates more accurate models • Can have a large number of design variables
Demerits	• Creates less accurate models • Has a limitation in the number of design variables	• May not find a global optimum design point
Iterative optimization method	Successive approximate optimization (SAO)	Approximation management framework (AMF)

Generally, shape optimization problems in aerospace engineering require many design variables. Since the local derivative-based optimization method can deal with such a larger number of design variables, this study focuses on the local derivative-based optimization method to be used for aerospace applications. The following chapter explains the AMF in detail and illustrates its technical barriers by using it to solve sample problems.

CHAPTER III

APPROXIMATE MANAGEMENT FRAMEWORK (AMF)

Developed by Alexandrov [6] in 1996, the AMF is an optimization framework using local derivative-based surrogate functions and the trust region model management strategy [21]. The optimum design point is iteratively updated until it satisfies user-defined criteria. This chapter begins with an overview of the AMF and then presents sample analytical problems solved using the AMF in order to reveal the technical barriers of the original AMF.

3.1 Overview of the Original AMF

A good figure that explains the original AMF can be found in Gano's dissertation [36]. A similar figure is shown in Figure 9 with slight modifications. The entire procedure of the original AMF follows:

Step 1: Evaluate the functions at the initial design point

At starting design point $\mathbf{x}_0$, the objective function and the constraint function are evaluated using both the high- and low-fidelity models.

Step 2: Evaluate the gradients at the current design point

At current design point $\mathbf{x}_n$, the gradient of the objective function and the constraint function are evaluated using both the high- and low-fidelity models. They are necessary not only for obtaining the gradients but for calculating approximate Hessian matrices. In the original AMF, derivatives are computed by means of the finite differentiation (FD) method.

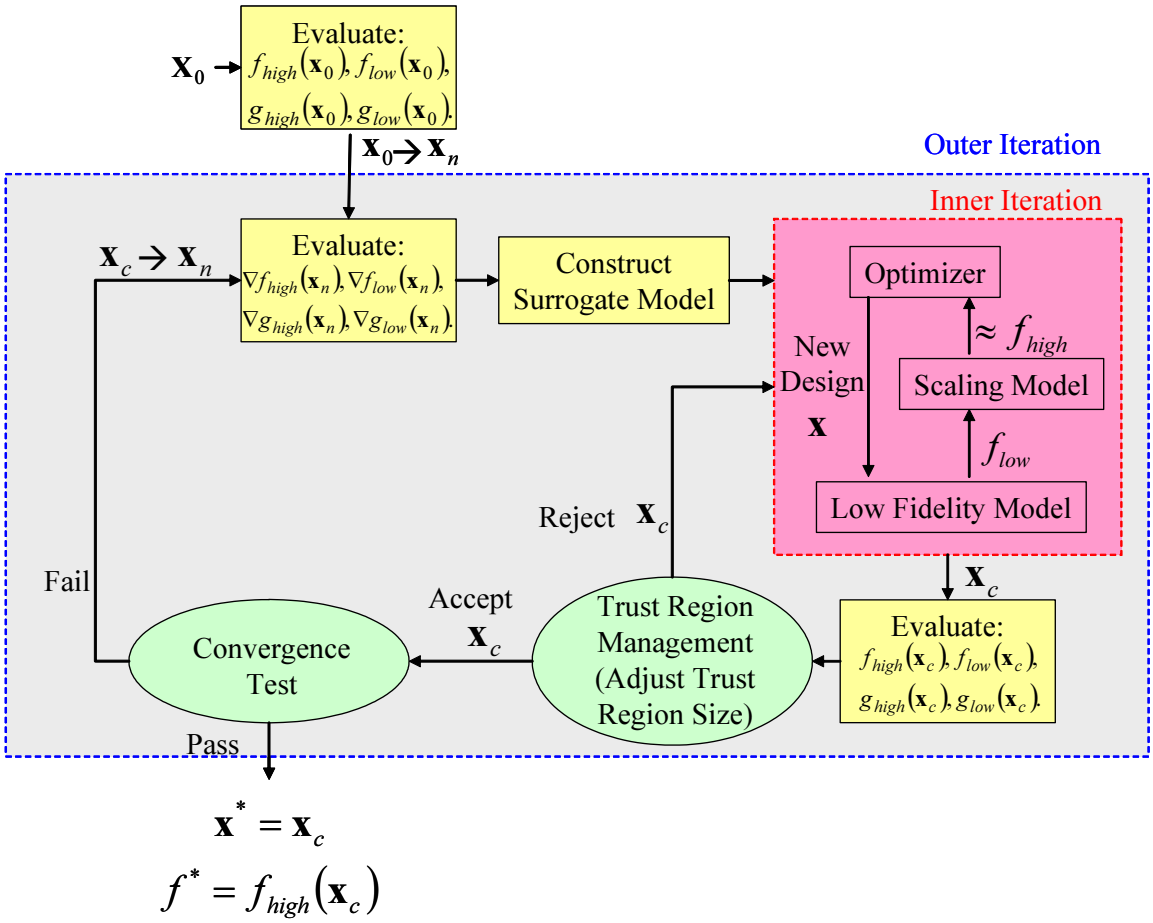


Figure 9: Overview of the AMF.

Step 3: Construct surrogate models

A scaling model that ensures matching among the different fidelity models is constructed. In conjunction with the low-fidelity model, the scaling model constitutes a surrogate model that can be based on either multiplicative or additive scaling. Each method can be modeled as first order, second order, or even higher order, depending on where the Taylor series is truncated. The surrogate model is constructed for the constraint function as well as for the objective function. In this study, the quasi-second-order additive scaling method is used in order to construct surrogate functions for the reasons mentioned in Chapter 2.

Step 4: Find an optimum design point using the surrogate model

With the surrogate model, which is composed of both the scaling and low-fidelity models, an optimum point is found within the trust region whose size is governed by the trust region management strategy explained in Step 6. The optimization problem solved in this step is described as follows:

$$\underset{\mathbf{x}}{\text{minimize}}: \quad f_{surrogate}(\mathbf{x}), \tag{17}$$

$$\text{subject to}: \quad \mathbf{g}_{surrogate}(\mathbf{x}) \leq \mathbf{0}, \tag{18}$$

$$\mathbf{g}_u(\mathbf{x}) \leq \mathbf{0}, \tag{19}$$

$$|x_i - x_{n,i}| \leq \frac{1}{2}|\Delta_{n,i}| \text{ for all } i, \tag{20}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}. \tag{21}$$

In Equation 19, subscript u of $\mathbf{g}_u$ indicates that the constraint is not scaled. An example of such non-scaled constraints is the thickness of the wing, a constraint that is solely based on the design variables (i.e., they are results neither from the low- nor the high-fidelity model). In Equation 20, x_i is the ith component of $\mathbf{x}$, $x_{n,i}$ is the ith component of $\mathbf{x}_n$, and $\Delta_{n,i}$ is the ith component of the trust region vector $\mathbf{\Delta}$ at the nth iteration. The constraint expressed in Equation 20 is necessary so that the optimization can be implemented in the trust region in which $\mathbf{x}_n$ is located at the center of the trust region, shown in Figure 10. In Equation 21, $\mathbf{l}$ and $\mathbf{u}$ are the lower and upper boundaries, respectively, for the design variable $\mathbf{x}$. The choice of the optimizer used here is based on preference. In work done by Alexandrov [7], three optimizers were compared: an augmented Lagrangian method [22], multilevel algorithms for a large-scale constrained optimization (MAESTRO) [8], and sequential quadratic programming (SQP) [13]. For typical single discipline problems, Alexandrov found the SQP to be the most promising, as does this research.

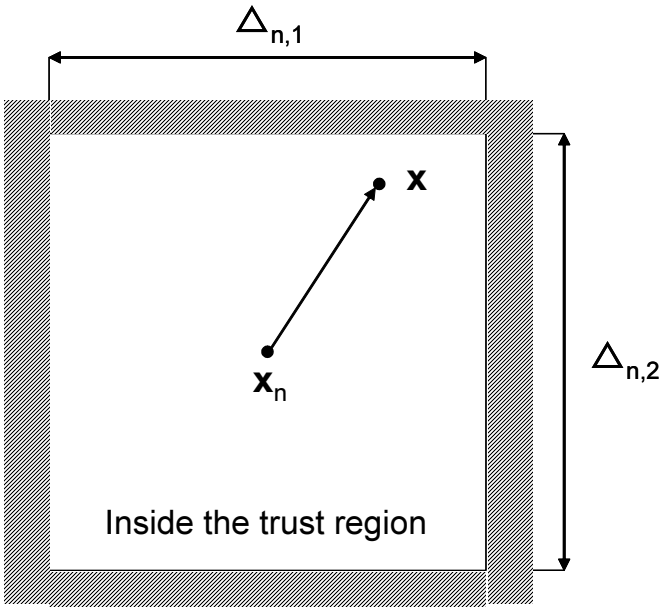


Figure 10: The trust region in a two-dimensional case.

Step 5: Evaluate a new design point and a penalty function

A new candidate design point $\mathbf{x}_c$ is found as a result of solving the optimization problem in Step 4. At this new candidate design point, the high- and low-fidelity objectives and the constraints are evaluated. These values are used to calculate a current value of the external penalty function $\phi(\mathbf{x_c})$, defined as follows:

$$\phi(\mathbf{x}) = f(\mathbf{x}) + r_p \sum_{i=1}^{m} \max[0,\, g_i(\mathbf{x})], \tag{22}$$

where r_p is the penalty weight, which is a relatively large number and typically increases by a factor of ten each time a new candidate design point is accepted, m is the number of constraints, and g_i is the ith component of the constraint. If the problem is unconstrained, note that $\phi(\mathbf{x})$ is just $f(\mathbf{x})$.

Step 6: Implement the trust region management

To help guarantee convergence of the variable fidelity optimization framework, the trust region model management strategy [21] was employed. This method provides a means for adaptively managing allowable move limits in the approximate design space. A trust region ratio allows the trust region model management framework to monitor how well the approximation matches the high-fidelity design space. Trust region ratio ρ_n (the suffix n is the current iteration counter) is the ratio of the actual change in the function to the predicted change of the function by the surrogate model, calculated at the new candidate design point $\mathbf{x}_c$ as follows:

$$\rho_n = \frac{\phi\left(\mathbf{x}_n\right)_{high} - \phi\left(\mathbf{x}_c\right)_{high}}{\phi\left(\mathbf{x}_n\right)_{surrogate} - \phi\left(\mathbf{x}_c\right)_{surrogate}}, \tag{23}$$

where

$$\phi\left(\mathbf{x}_n\right)_{high} = f_{high}\left(\mathbf{x}_n\right) + r_p \sum_{i=1}^{m} \max\left[0,\, g_{high,i}\left(\mathbf{x}_n\right)\right], \tag{24}$$

$$\phi\left(\mathbf{x}_c\right)_{high} = f_{high}\left(\mathbf{x}_c\right) + r_p \sum_{i=1}^{m} \max\left[0,\, g_{high,i}\left(\mathbf{x}_c\right)\right], \tag{25}$$

$$\phi\left(\mathbf{x}_n\right)_{surrogate} = f_{surrogate}\left(\mathbf{x}_n\right) + r_p \sum_{i=1}^{m} \max\left[0,\, g_{surrogate,i}\left(\mathbf{x}_n\right)\right], \tag{26}$$

$$\phi\left(\mathbf{x}_c\right)_{surrogate} = f_{surrogate}\left(\mathbf{x}_c\right) + r_p \sum_{i=1}^{m} \max\left[0,\, g_{surrogate,i}\left(\mathbf{x}_c\right)\right]. \tag{27}$$

Note that by definition, $\phi\left(\mathbf{x}_n\right)_{high} = \phi\left(\mathbf{x}_n\right)_{surrogate}$ because the surrogate model matches the high-fidelity model at $\mathbf{x}_n$. After the trust region ratio is calculated, the trust region size is computed by the following rules [68]:

$$\boldsymbol{\Delta}_{n+1} = \begin{cases} 0.25\boldsymbol{\Delta}_n & : \quad \rho_n \leq 0.25,\ 1.25 \leq \rho_n \\ \boldsymbol{\Delta}_n & : \quad 0.25 \leq \rho_n \leq 0.75 \\ \Gamma\boldsymbol{\Delta}_n & : \quad 0.75 \leq \rho_n \leq 1.25 \end{cases} . \tag{28}$$

Here $\Gamma = 3$ if any i satisfies $|\, x_{i,c} - x_{i,n} \,| = \frac{1}{2} |\Delta_{i,n}|$ (i.e., the new candidate point is on the edge of the trust region); otherwise, $\Gamma = 1$. Physically, ρ_n represents how good a surrogate model is compared to the high-fidelity model. If ρ_n is between 0.75 and 1.25, the approximation is excellent, and the size of the trust region may increase, depending on where the candidate design point $\mathbf{x}_c$ is. If ρ_n is between 0.25 and 0.75, then the approximation is good, so the size of the trust region remains. If ρ_n is between 0 and 0.25 or greater than 1.25, then the approximation still captures the trend of the high-fidelity model, but it is not as good, so the size of the trust region shrinks. If ρ_n is negative, the candidate design point $\mathbf{x}_c$ is a worse design point because the sign of the actual change in the function is opposite that of the predicted change of the function by the surrogate model. In this case, the candidate design point is rejected, the trust region size is reduced by 0.25, and the algorithm returns to Step 4 (i.e., an optimization is executed again with the same setting as the previous iteration, while the size of the trust region shrinks). As long as $\rho_n > 0$, the point is accepted, and the algorithm proceeds to Step 7. The relationship between the trust region ratio and the trust region size is depicted in Figure 11.

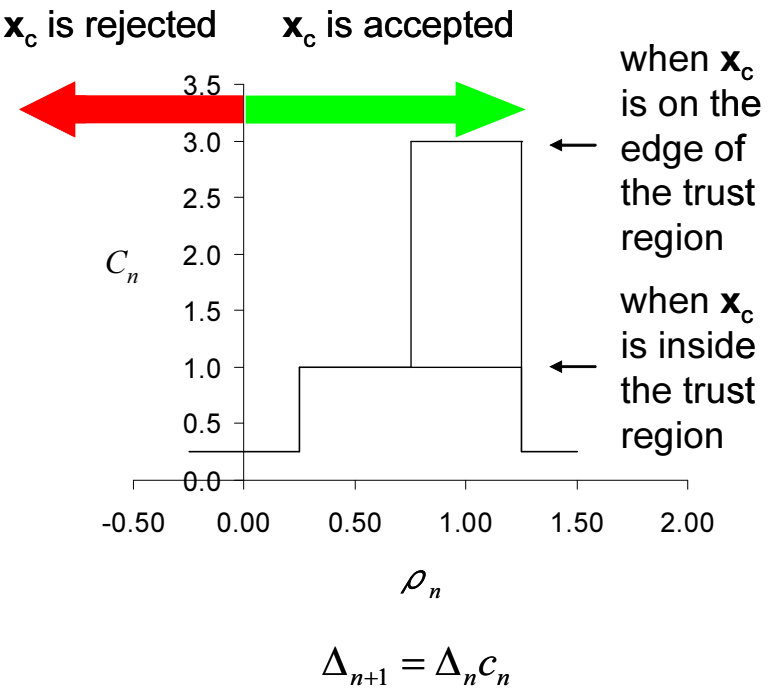


Figure 11: The relationship between the trust region and the trust region size.

Step 7: Test convergence

For the implementation used in this research, convergence is determined by the following stopping criteria:

$$| f_{high}(\mathbf{x}_{n+1}) - f_{high}(\mathbf{x}_n)| < \varepsilon_f, \tag{29}$$

$$\|\mathbf{x}_{n+1} - \mathbf{x}_n\| < \varepsilon_{\mathbf{x}}, \tag{30}$$

where ε_f and $\varepsilon_{\mathbf{x}}$ are tolerances depending on the problem and n is the current iteration counter. If either of the two inequalities is satisfied, the algorithm is considered to have converged. If the convergence test is true, the final design is found; otherwise, the algorithm returns to Step 2.

3.2 Implementation of the Original AMF

In order to investigate the capability of the original AMF, simple analytical problems are solved using the original AMF.

3.2.1 Noisy Problem

Generally, numerical noise in computational models tends to be more conspicuous as their fidelity increases. Therefore, solving a simple analytical noisy problem by the original AMF, as Marduel [55] did, is valuable.

$$\underset{\mathbf{x}}{\text{minimize}} \quad : \quad f_{high} = \left[100\left(x_1 - x_2^2\right)^2 + \left(1 - x_2\right)^2\right]\left(1 + n\left(x_1,\, x_2\right)\right), \tag{31}$$

$$f_{low} = \left[100\left(x_1 + 0.1 - x_2^2\right)^2 + \left(1.5 - x_2\right)^2\right]\left(1 + n\left(x_1,\, x_2\right)\right), \tag{32}$$

where $n\left(x_1,\, x_2\right)$ represents the noise function given by

$$\begin{aligned} n\left(x_1,\, x_2\right) \;=\; & A\left(\sin\left(w_1\left(2x_1 + x_2\right)\right) + \cos\left(w_2\left(x_1 + 2x_2\right)\right)\right. \\ & - \left|\cos\left(w_3\left(2x_1 + x_2\right)\right)\sin\left(w_4\left(x_1 + 2x_2\right)\right)\right| \\ & \left. + \left|\sin\left(w_5\left(2x_1 + x_2\right)\right)\right|\right), \end{aligned} \tag{33}$$

where $w_i s, (w = 1, ..., 5)$ are very high-frequency pulsations ($\approx 10^9$), and A is the amplitude of the pulsations (the value of A is not specified in reference [55]). If $n\left(x_1,\, x_2\right)$ is set to zero in Equation 31, the equation is referred to as "the Rosenbrock function" or "the Rosenbrock's banana function" [74], depicted in Figure 12, which has the global minimum zero at $(x_1, x_2) = (1.0, 1.0)$. When computing derivatives, Marduel used a forward finite differentiation whose step size was 10^{-2} and investigated the effect of numerical noise on the original AMF.

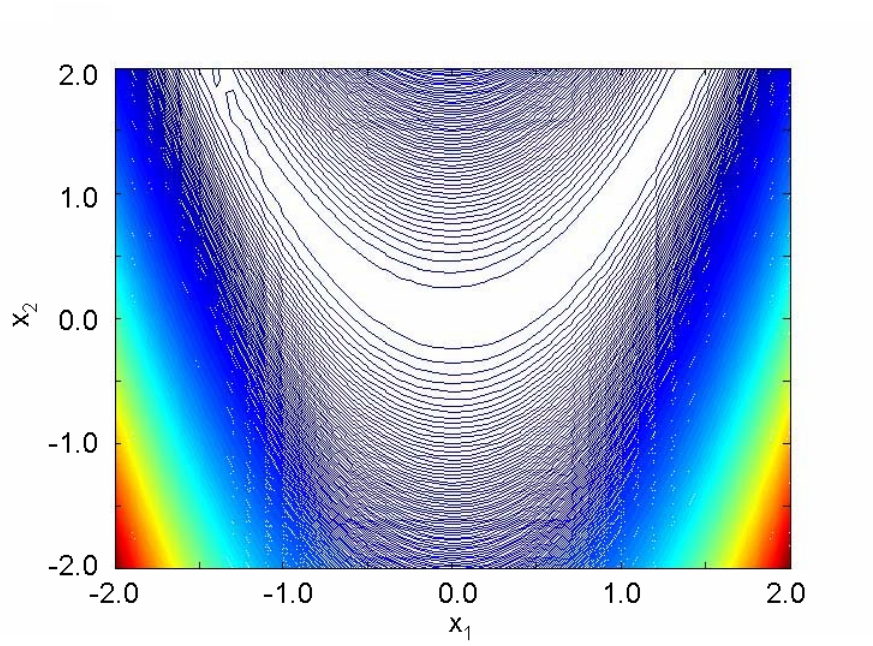


Figure 12: The Rosenbrock function.

Marduel's results are summarized in Table 2, where the values indicate the final high-fidelity objective function values and the values in parentheses indicate the number of high-fidelity function calls for computing gradients.

Table 2: The final function values and the number of the high-fidelity function calls for computing gradients in the noisy problem [55].

	Surrogate function type in the AMF		Direct optimization
	Additive 1st order	Additive 2nd order	
Without noise	8.7E-7 (28)	3.8E-4 (4)	3.8E-4 (30)
With noise	2.2E-1 (8)	1.0 (3)	1.4E-4 (37)

Table 2, as Marduel pointed out, shows that noise contributes more to deterioration of the original AMF than direct optimization. Although the convergence criteria are met, the optimization with the original AMF is terminated prematurely

due to erroneous derivatives computed under the noise. Therefore, instead of finite differentiation, new techniques that can compute correct derivatives, even in a noisy environment, should be introduced to the original AMF so that it is more efficient and robust.

3.2.2 Constrained Problem

Since most engineering problems have constraints, solving a simple analytical constraint problem by the AMF is valuable. Thus, using the AMF, this research will attempt to solve the following problem, expressed by Equations 34 to 39 and depicted in Figure 13. This problem has a global minimum of 5.6684 at $(x_1, x_2) = (0.8842, 1.1507)$. Note that derivatives are analytically computed in order to remove any effects related to numerical noise, discussed in 3.2.1 in this case.

$$\begin{aligned}
\underset{\mathbf{x}}{\text{minimize}} \quad &: \quad f_{high} = 4x_1^2 + x_2^3 + x_1 x_2, && (34)\\
\text{subject to} \quad &: \quad g_{high} = \frac{1}{x_1} + \frac{1}{x_2} - 2 \leq 0, && (35)\\
&\quad\; 0.1 < x_1 < 10.0, && (36)\\
&\quad\; 0.1 < x_2 < 10.0, && (37)\\
&\quad\; f_{low} = 4\left(x_1 + 0.1\right)^2 + \left(x_2 - 0.1\right)^3 + x_1 x_2 + 0.1, && (38)\\
&\quad\; g_{high} = \frac{1}{x_1} + \frac{1}{x_2 + 0.1} - 2 - 0.001 \leq 0. && (39)
\end{aligned}$$

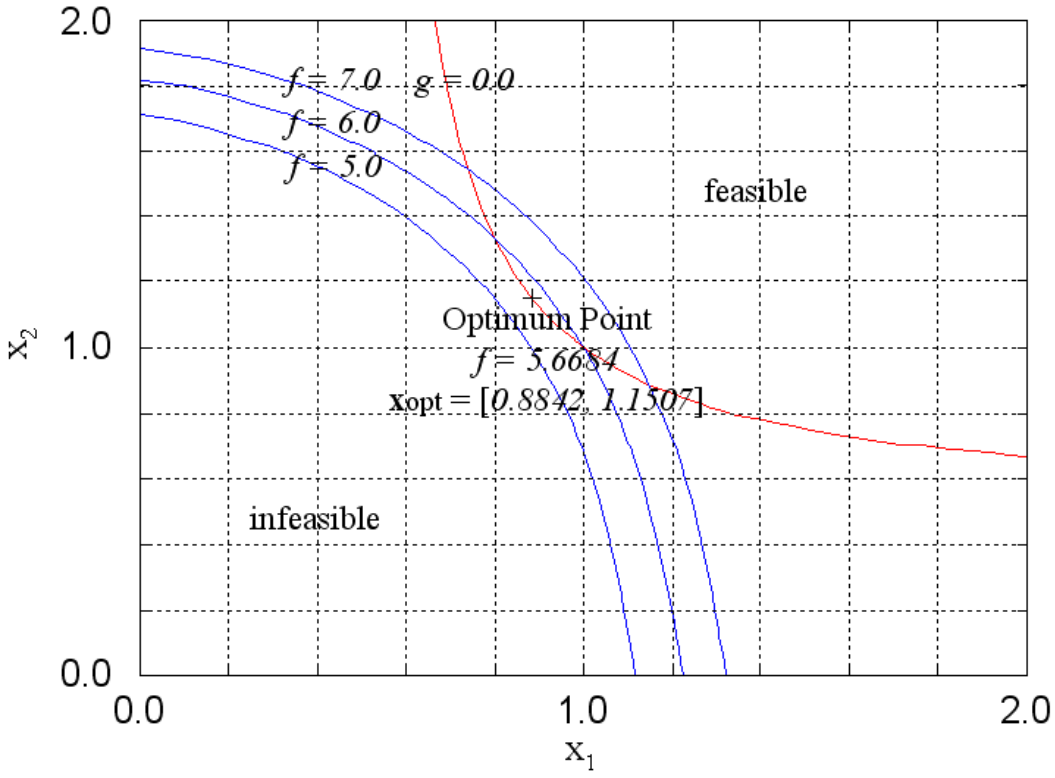


Figure 13: Analytical constrained problem.

In the original AMF, the location of an initial point and the size of an initial trust region are user dependent. When the size of the initial trust region is the entire domain, Figure 14 shows relationships between initial points and corresponding final solutions. Figures 15 and 16 illustrates the relationships between the initial points and the corresponding numbers of the high- and low-fidelity function calls, respectively, when the size of the initial trust region is the entire domain. Here, 1.0e-4 is used for the criteria in Equations 29 and 30. Note that the maximum number of high-fidelity function calls is limited to 100 in this optimization problem. Although Figures 14, 15, and 16 are obtained when the size of the initial trust region is 100% of the entire domain, similar tendencies can be observed in any sizes of the initial trust region. As references, results from a direct optimization with SQP, whose derivatives are

computed analytically and whose convergence criteria corresponding to Equations 29 and 30 are 1.0e-4, are shown in Figures 17 and 18.

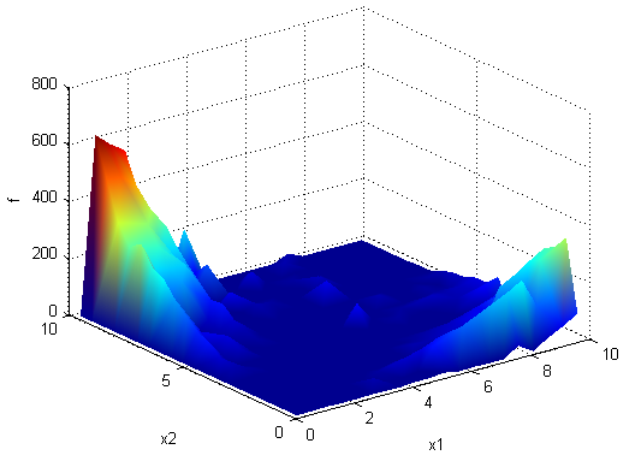


Figure 14: Relationships between initial points and corresponding final solutions (original AMF).

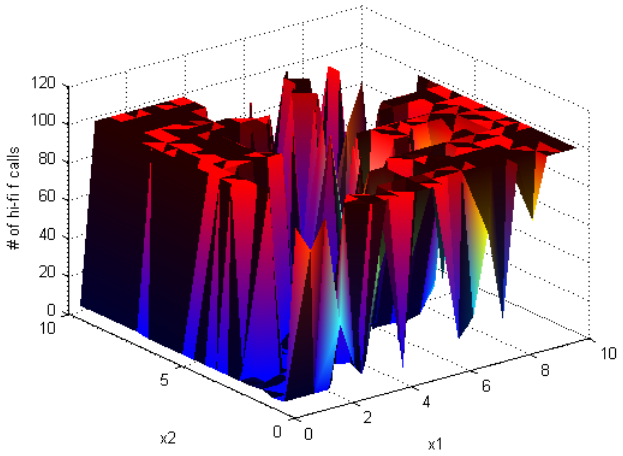


Figure 15: Relationships between initial points and corresponding numbers of high-fidelity function calls (original AMF).

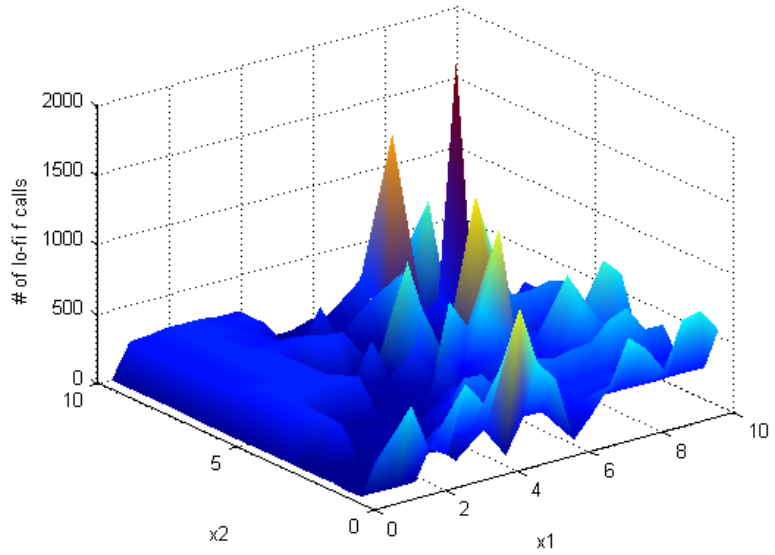


Figure 16: Relationships between initial points and corresponding numbers of low-fidelity function calls (original AMF).

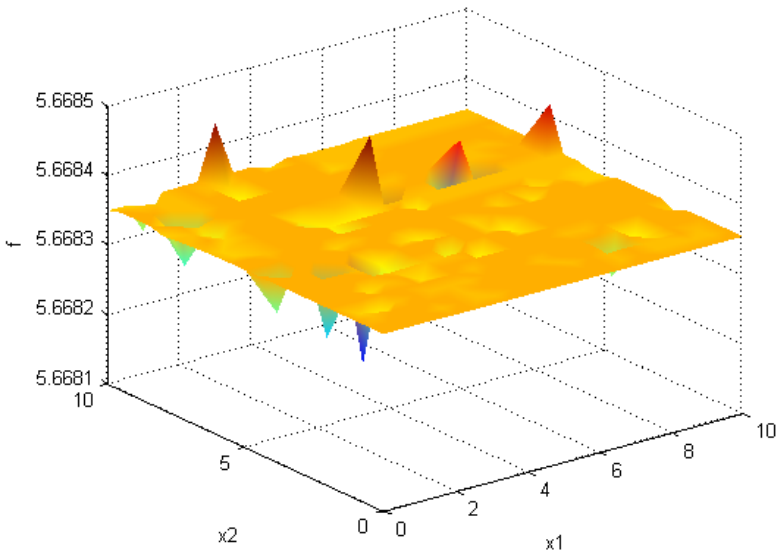


Figure 17: Relationships between initial points and corresponding converged solutions (SQP).

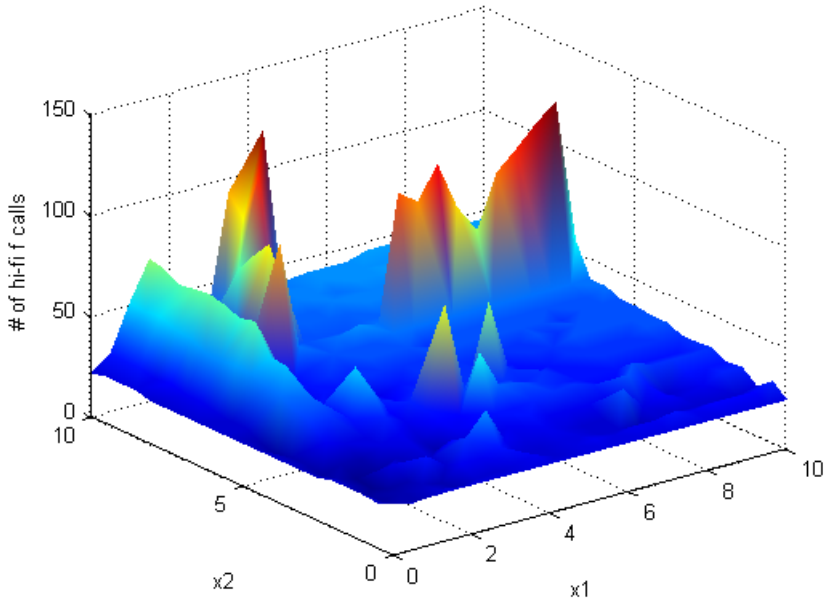


Figure 18: Relationships between initial points and corresponding numbers of the high-fidelity function calls (SQP).

Figure 14 shows that whether or not a correct solution can be obtained depends on the initial point, and often the original AMF fails to find the true optimum point. In addition, Figures 15 and 16 show that the numbers of high- and low-fidelity function calls tend to be very large when solutions do not denote a true optimum.

As an example of successful cases, a result whose initial point is at $(1.5, 1.5)$ is shown in Table 3, in which a result with SQP is also shown as a reference. From Table 3, the effectiveness of the original AMF is clearly seen since the number of high-fidelity function calls with the original AMF is 33% of that with the SQP and since the final optimum point with the original AMF is very close to that with the SQP.

Table 3: Results for the constrained problem with the initial point at $(1.5, 1.5)$

	AMF	SQP
# of hi-fi function calls	6	18
# of lo-fi function calls	51	-
Final hi-fi function	5.6683	5.6684
Final x_1	0.8823	0.8842
Final x_2	1.1538	1.1507

.

As an example of failed cases, a result whose initial point is at $(8.0, 1.0)$ is shown in Table 4, in which a result with SQP is also shown as a reference. Table 4 shows that the number of high- and low-fidelity function calls with the original AMF is much larger than that with the SQP and that the final point with the original AMF is far different from that with the SQP. It should be noted that the AMF case did not converged but was terminated by the user specified limit of 100 high-fidelity function calls.

Table 4: Results for the constrained problem with the initial point at $(8.0, 1.0)$.

	AMF	SQP
# of hi-fi function calls	100	22
# of lo-fi function calls	300	-
Final hi-fi function	222.4729	5.6684
Final x_1	7.3927	0.8842
Final x_2	0.5363	1.1507

Figure 19 and Table 5 summarize the history of this case. As can be seen in Table 5, the trust region ratio sometimes becomes a large negative number. From the equation

for the trust region ratio shown in Equations 23, 24, 25, 26, and 27, one can infer that the large negative number results from a violation against the high-fidelity constraint at the new point $\mathbf{x_c}$ even though the surrogate constraint is satisfied within some tolerance at the same point. However, this is very natural because surrogate models based on the Taylor series deteriorate as the distance from base point $\mathbf{x_n}$ becomes large, as shown in Figure 20. Then, this large negative trust region ratio causes the size of the trust region to shrink and limits the movement of the new point $\mathbf{x_c}$. As a result of the many rejections caused by the large negative trust region ratio, the final point becomes stuck in an inaccurate location, as shown in Figure 19.

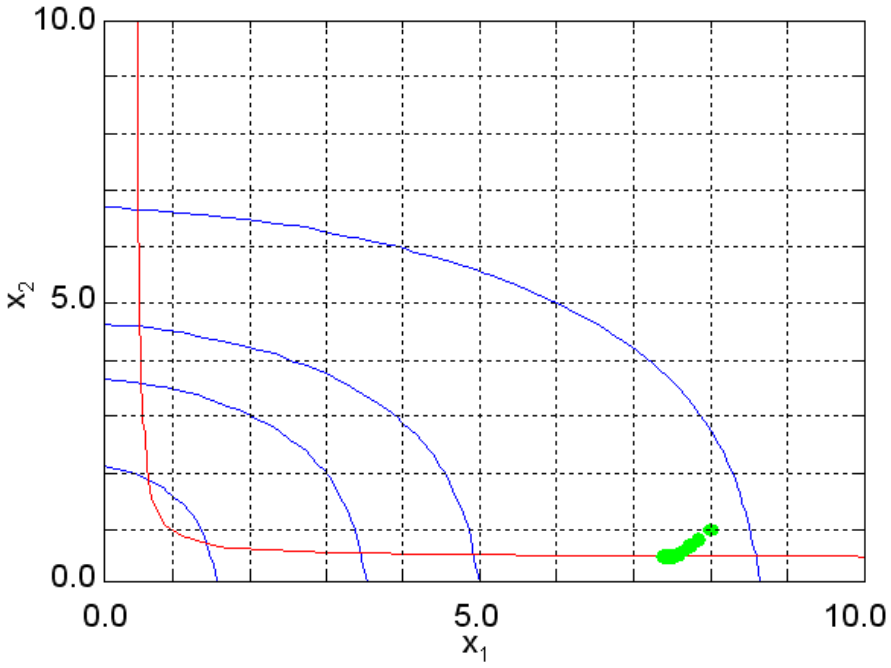


Figure 19: Visual history for the constrained problem with an initial point at $(8.0, 1.0)$.

Table 5: Numerical history for the constrained problem with an initial point at $(8.0, 1.0)$.

Iter	$\mathbf{x}_n$		$\mathbf{x}_c$		ρ_n	g_{high}	$g_{surrogate}$	$\mathbf{x}_c$
n	x_1	x_2	x_1	x_2				accepted?
1	8.0000	1.0000	0.8690	1.1695	-1.2632	5.8652E-03	3.8559E-06	No
2	8.0000	1.0000	5.5250	0.5087	-103.8246	1.4677E-01	1.4101E-07	No
3	8.0000	1.0000	7.3812	0.4932	-384.0348	1.6291E-01	1.0773E-05	No
4	7.8453	0.8453	7.8453	0.8453	0.9994	-6.8954E-01	-6.9693E-01	Yes
5	7.8453	0.8453	7.3813	0.5001	-4254.5475	1.3498E-01	9.6134E-10	No
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
100	7.3884	0.5363	7.3884	0.5363	1.0000	-4.3743e-014	6.6613e-016	Yes

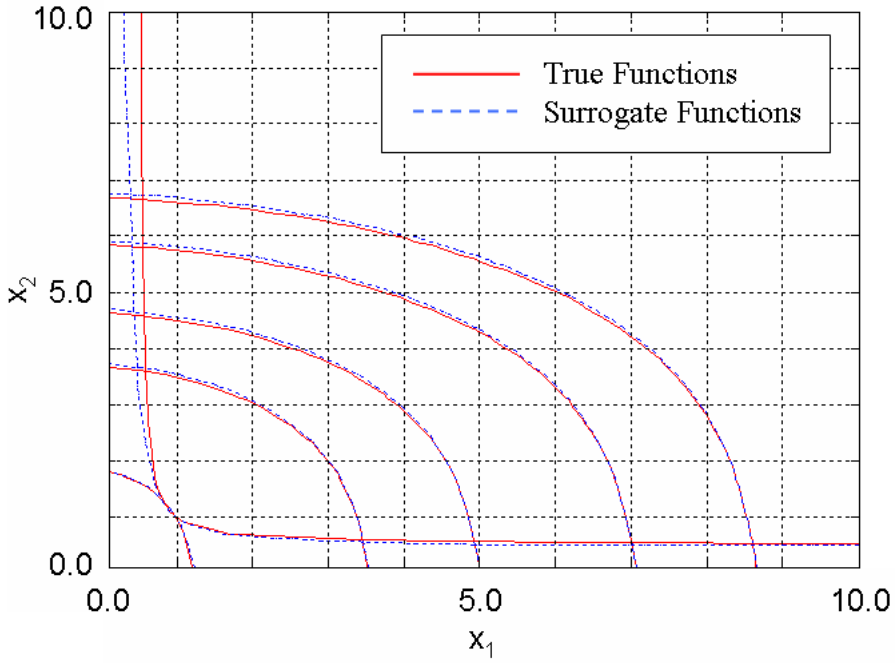


Figure 20: True functions and surrogate functions whose base point $\mathbf{x}_n$ is (8.0. 1.0).

3.3 Technical Barriers of the Original AMF

The implementations of the original AMF conducted in Section 3.2 revealed the following technical barriers.

Technical barrier 1

Computing derivatives by means of the finite differentiation technique often degrades the original AMF due to the noise of the computational models.

Technical barrier 2

When constraints exist in optimization problems, the original AMF often fails due to the inappropriate treatment of the trust region ratio.

Consequently, the following research questions emerge:

Research question 1

How can one obtain correct derivatives, even in noisy functions, in a shorter time?

Research question 2

The governing equation of the trust region ratio does not work properly when an optimization problem has constraints, leading to inaccurate behavior in the original AMF. How can one modify the governing equation of the trust region ratio even if the optimization problem has constraints?

In an attempt to answer these research questions, the next chapter presents several hypotheses that pertain to the original AMF.

CHAPTER IV

ROBUST AMF

In order to create a more robust AMF, this chapter develops and presents several new techniques.

4.1 Techniques for Computing Derivatives

In optimization techniques based on the gradient descent method, calculating gradients accurately in a short time period is very important. Currently, derivatives are calculated by three well-known mathematical methods: the finite differentiation (FD) method, the adjoint method, and the automatic differentiation (AD) method. In the following subsections, these methods will be explained in detail and then summarized in Table 7 at the end of this section.

4.1.1 The Finite Differentiation (FD) Method

Among the methods used to calculate derivatives, the oldest and simplest method is the FD method. This method requires function values at two adjacent points and calculates derivatives by dividing the difference of the function values by the difference between the two adjacent points. The FD chooses the two points in one of three ways: forward difference, backward difference, or central difference. The FD with forward difference is mathematically expressed as follows:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_i} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{f(\mathbf{x}+\Delta h\mathbf{a}_1)-f(\mathbf{x})}{\Delta h} \\ \vdots \\ \frac{f(\mathbf{x}+\Delta h\mathbf{a}_i)-f(\mathbf{x})}{\Delta h} \\ \vdots \\ \frac{f(\mathbf{x}+\Delta h\mathbf{a}_n)-f(\mathbf{x})}{\Delta h} \end{pmatrix}, \tag{40}$$

where $\mathbf{x}$ is composed of n components, Δh is a user-dependent small number, and $\mathbf{a}_i$ is a vector whose ith component is one and others are all zeros. The FD method can easily be implemented for computing derivatives because one can completely treat an analytical tool as a black box. However, finding a proper size of Δh is sometimes very difficult. Indeed, depicted in Figure 21, the output of analytical tools, particular higher-fidelity tools such as CFD, usually have numerical noise, and derivatives computed by the FD method are often not reliable even if the size of Δh is very small. In addition, the computation of $\nabla f(\mathbf{x})$ requires $n+1$ function evaluations, which are very inefficient when n is a large number and when a function call is expensive. Thus, the FD method, if possible, should not be used in optimizations based on the gradient descent method.

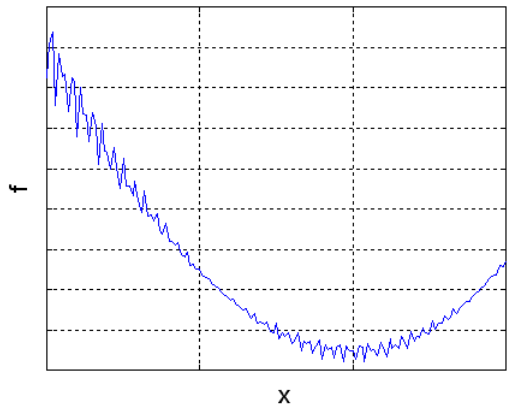


Figure 21: Notional picture of a noisy function.

4.1.2 The Adjoint Method

The adjoint method, sometimes referred to as "control theory," was developed by Jameson [48]. This method is specialized for computing derivatives in CFD analyses. Since objective function I (e.g., C_D, C_L) is a function of the flow-field variables $\mathbf{w}$ and the physical location of boundary $\mathbf{S}$, it can be expressed as follows:

$$I = I\left(\mathbf{w},\, \mathbf{S}\right). \tag{41}$$

Then, the sensitivity derivative of the objective function I is given as follows:

$$\delta I = \left(\frac{\partial I}{\partial \mathbf{w}}\right)^T \delta\mathbf{w} + \left(\frac{\partial I}{\partial \mathbf{S}}\right)^T \delta\mathbf{S}. \tag{42}$$

For the same flow field, the governing flow equation R, such as Navier-Stokes or Euler equations, satisfies the following condition:

$$R\left(\mathbf{w},\, \mathbf{S}\right) = 0. \tag{43}$$

In steady flow conditions, the total derivative of the governing flow equation is null because the flow condition does not change. Therefore, the total derivative of the governing flow equation becomes

$$\delta R = \left(\frac{\partial R}{\partial \mathbf{w}}\right) \delta\mathbf{w} + \left(\frac{\partial R}{\partial \mathbf{S}}\right) \delta\mathbf{S} = 0. \tag{44}$$

In such a case, one can combine Equations 42 and 44 by introducing a Lagrange multiplier ψ and rewrite the sensitivity derivative of the objective function I as follows:

$$\begin{aligned} \delta I &= \left(\frac{\partial I}{\partial \mathbf{w}}\right)^T \delta\mathbf{w} + \left(\frac{\partial I}{\partial \mathbf{S}}\right)^T \delta\mathbf{S} - \psi^T \left[\left(\frac{\partial R}{\partial \mathbf{w}}\right) \delta\mathbf{w} + \left(\frac{\partial R}{\partial \mathbf{S}}\right) \delta\mathbf{S}\right] && (45) \\ &= \left[\left(\frac{\partial I}{\partial \mathbf{w}}\right)^T - \psi^T \left(\frac{\partial R}{\partial \mathbf{w}}\right)\right] \delta\mathbf{w} + \left[\left(\frac{\partial I}{\partial \mathbf{S}}\right)^T - \psi^T \left(\frac{\partial R}{\partial \mathbf{S}}\right)\right] \delta\mathbf{S}. && (46) \end{aligned}$$

If the equation inside the brackets of the first term in Equation 46 is set to zero, it is referred to as an "adjoint equation." If one can solve the adjoint equation, the first term of Equation 46 is eliminated, and the sensitivity derivative of the objective function I becomes

$$\delta I = \left[\left(\frac{\partial I}{\partial \mathbf{S}} \right)^T - \psi^T \left(\frac{\partial R}{\partial \mathbf{S}} \right) \right] \delta \mathbf{S}. \tag{47}$$

Now, Equation 47 is independent of flow-field variables $\mathbf{w}$, indicating that solving the adjoint equation as well as the governing flow equation is sufficient for obtaining the sensitivity derivative of the objective function I.

As explained above, the adjoint method is a very elegant method for computing derivatives. However, one can not treat an analytical tool as a "black box" anymore since the adjoint method requires a significant computational and mathematical pre-processing before its application. Therefore, even many years after its birth, only certain research groups continue to use the adjoint method.

4.1.3 Automatic Differentiation (AD) Method

The history of the AD method can be traced back to the 1960s [84]. The AD method exploits the fact that any arbitrarily complex function, when programmed into a digital computer, can be expressed by a series of basic arithmetic operations (e.g., additions and multiplications) and basic functions (e.g., sin and exp). Derivatives are computed by repeatedly applying the chain rule of differentiation following a set of prescribed arithmetic rules. Depending on the arithmetic rules, the AD method can be classified into two methods: the forward automatic differentiation (FAD) method and the reverse automatic differentiation (RAD) method. When to use each of them depends on the numbers of input and output variables involved. In 4.1.3.1 and 4.1.3.2, these methods are explained in detail with simple examples.

4.1.3.1 FAD

The arithmetic rule for the FAD is shown in Figure 22, and examples of computing a derivative by means of the FAD is presented in Figure 23. The name "forward" is derived from the fact that the direction in which the derivatives is are computed is identical to that in which the function value is computed. That is, the derivatives and the function value can be computed simultaneously. The computational cost of the FAD is proportional to the number of input design variables and independent of the number of output functions. Therefore, if the number of output functions is greater than that of the input design variables, the FAD is a better method for computing derivatives than the RAD.

Figure 24 shows a Fortran subroutine program implementing the computations shown in Figure 23. As explained, this Fortran subroutine program should be called twice in order to compute $\frac{\partial f}{\partial x_1}$ $(d_1 = 1.0, d_2 = 0.0)$ and $\frac{\partial f}{\partial x_2}$ $(d_1 = 0.0, d_2 = 1.0)$.

Suppose an output function f is expressed by n input variables, $x_1, ..., x_n$. In order to calculate a derivative of the output function f with respect to the ith input variable x_i, FAD implements the following procedure:

1. Introduce t_js ($t_j = x_j, j = 1, ..., n$) and decompose the objective function f by using t_ks ($k = 1, ..., m$). Name the intermediate functions ψ_ls ($l = n+1, ..., m$).

2. Introduce d_j ($j = 1, ..., n$) and initialize them as $d_i = 1$, $d_p\,(p \neq i) = 0$.

3. Introduce d_l ($l = n+1, ..., m$) and compute them as $d_l = \sum \left(\frac{\partial \psi_l}{\partial t_q}\right) d_q$, where t_qs are terms used in the intermediate function ψ_l.

4. Set $\frac{\partial f}{\partial x_i} = d_m$.

Note that if x_as ($a = 1, ..., n$) are controlled by β_bs ($b = 1, ..., p$) and computing $\frac{\partial f}{\partial \beta_q}$ ($1 \leq q \leq p$) is desired, $d_s = \frac{\partial x_s}{\partial \beta_q}$ ($s = 1, ..., n$) in Step 2.

Figure 22: The arithmetic rule of the FAD.

$f(x_1, x_2) = x_1 + x_2 \cos x_1 x_2$. What is $\frac{\partial f}{\partial x_1}$?

$t_1 = x_1$

$t_2 = x_2$

$t_3 = t_1 t_2 \equiv \psi_3$

$t_4 = \cos t_3 \equiv \psi_4$

$t_5 = t_2 t_4 \equiv \psi_5$

$t_6 = t_1 + t_5 \equiv \psi_6$

$d_1 = 1$

$d_2 = 0$

$d_3 = \sum (\partial \psi_3 / \partial t_q)\, d_q = t_2 d_1 + t_1 d_2 = t_2$

$d_4 = \sum (\partial \psi_4 / \partial t_q)\, d_q = (-\sin t_3)\, d_3 = -t_2 \sin t_3$

$d_5 = \sum (\partial \psi_5 / \partial t_q)\, d_q = t_4 d_2 + t_2 d_4 = -t_2^2 \sin t_3$

$d_6 = \sum (\partial \psi_6 / \partial t_q)\, d_q = d_1 + d_5 = 1 - t_2^2 \sin t_3$

$\partial f / \partial x_1 = 1 - t_2^2 \sin t_3$

If $x_1 = \beta_1 + \beta_2, x_2 = 2\beta_1 + \beta_2$ in the problem above, what is $\frac{\partial f}{\partial \beta_1}$?

$d_1 = \frac{\partial x_1}{\partial \beta_1} = 1$

$d_2 = \frac{\partial x_2}{\partial \beta_1} = 2$

$d_3 = \sum (\partial \psi_3 / \partial t_q)\, d_q = t_2 d_1 + t_1 d_2 = t_2 + 2t_1$

$d_4 = \sum (\partial \psi_4 / \partial t_q)\, d_q = (-\sin t_3)\, d_3 = (-\sin t_3)\, (t_2 + 2t_1)$

$d_5 = \sum (\partial \psi_5 / \partial t_q)\, d_q = t_4 d_2 + t_2 d_4 = 2t_4 - t_2(t_2 + 2t_1) \sin t_3$

$d_6 = \sum (\partial \psi_6 / \partial t_q)\, d_q = d_1 + d_5 = 1 + 2t_4 - t_2(t_2 + 2t_1) \sin t_3$

$\partial f / \partial \beta_1 = 1 + 2\cos t_3 - x_2(x_2 + 2x_1) \sin x_1 x_2$

Figure 23: Examples of the FAD.

```
1       SUBROUTINE FUNC_D(t1, d1, t2, d2, y, yd)
2       IMPLICIT NONE
3       REAL t1, d1, t2, d2, y, yd
4       INTRINSIC COS
5       yd = d1 + d2*COS(t1*t2) - t2*(d1*t2+t1*d2)*SIN(t1*t2)
6       y = t1 + t2*COS(t1*t2)
7       RETURN
8       END
```

Figure 24: A derivative code implementing the procedure in Figure 23.

4.1.3.2 RAD

The arithmetic rule for the RAD and an example of computing derivatives by means of the RAD are presented in Figures 25 and 26. In Figures 25 and 26, $\bar{t}$s, which are referred to as "adjoint variables," are newly introduced. Therefore, computer codes generated through the RAD are often referred to as "adjoint codes," and the author follows the same convention in this study as well. In the RAD, a two-staged process is implemented: one that is surrounded by dotted lines in Figure 26 and whose direction is identical to that of computing functions, and the other that is surrounded by solid lines in Figure 26 and whose direction is opposite to the former. The term "reverse" is derived from the fact that the direction in which the derivatives are computed is opposite that in which the functions are computed. Some intermediate results computed in the first path are stored in memory in order to recall them in the second path. The computational cost of the RAD is proportional to the number of the output functions and independent of the number of input design variables. Therefore, if the number of input design variables is greater than that of output functions, the RAD may be a better method to compute derivatives than the FAD.

Figure 27 shows a Fortran subroutine program implementing the computations shown in Figure 26. As explained, this Fortran subroutine program, which stores results in the first path in "tempb (line 7) " is called only once in order to compute $\frac{\partial f}{\partial x_1}$ and $\frac{\partial f}{\partial x_2}$.

Suppose an output function f is expressed by n input variables, $x_1, ..., x_n$. In order to calculate a derivative of the output function f with respect to the ith input variable x_i, RAD implements the following procedure:

1. Decompose the objective function f into intermediate functions ψ_ls $(l = n + 1, ..., m)$.

2. Calculate ψ_ls $(l = n + 1, ..., m)$ and memorize the results.

3. Introduce adjoint variables $\bar{t}_k$s $(k = 1, ..., m)$ and initialize them as $\bar{t}_j = 0$ $(j = 1, ..., m - 1)$, $\bar{t}_m = 1$.

4. Starting from $l = m$ and ending $l = n + 1$, calculate $\bar{t}_q$s as $\bar{t}_q = \bar{t}_q + \left(\frac{\partial \psi_i}{\partial t_q}\right) \bar{t}_i$, where t_qs are terms used in the intermediate function ψ_l.

5. Set $\frac{\partial f}{\partial x_i} = \bar{t}_i \, (i = 1, ..., n)$.

Figure 25: The arithmetic rule of the RAD.

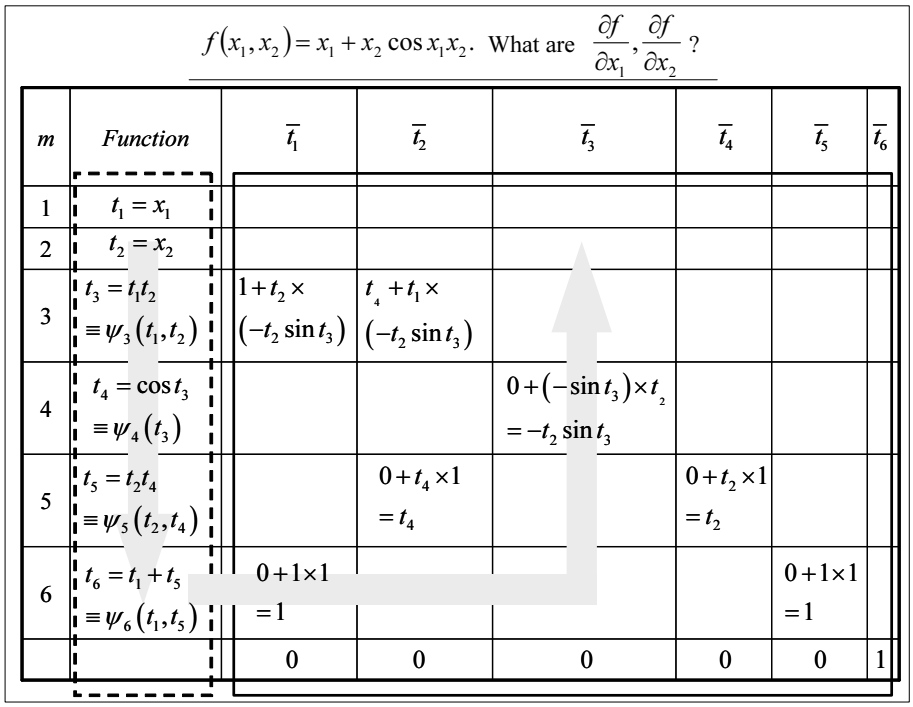


Figure 26: An example of the RAD.

```
1   SUBROUTINE FUNC_B(t1, t1b, t2, t2b, y, yb)
2   IMPLICIT NONE
3   REAL t1, t1b, t2, t2b, y, yb
4   REAL tempb
5   INTRINSIC COS
6
7   tempb = -(t2*SIN(t1*t2)*yb)
8   t1b = t2*tempb + yb
9   t2b = t1*tempb + COS(t1*t2)*yb
10  yb = 0.0
11
12  RETURN
13  END
```

Figure 27: A derivative code implementing the procedure in Figure 26.

4.1.3.3 AD tools

Computer codes are regarded as a series of functions that can be differentiated by means of AD tools. They read the original computer code for computing functions and generate new computer codes for calculating the derivatives of the function with respect to specified variables by following the arithmetic rules of the AD method. Although several AD tools are available, their readable languages, which are summarized in Table 6, are limited.

Table 6: Representative AD tools and their readable languages.

ADIFOR [78]	Fortran77
OpenAD [60]	Fortran77, Fortran90
TAPENADE [46]	Fortran77, Fortran90, Fortran95
ADIC [12]	ANSI C
ADMC++ [23]	C++, MATLAB

In this study, TAPENADE is used because it is the easiest tool to use. One important pre-processing task in TAPENADE is the reconstruction of an original source code. As depicted in Figure 28, if input variables are read and a function value is computed in the same program in the original source code, one should divide it into a main program and a set of subroutine codes whose primary code is referred to as a "top routine" in order to generate a derivative code through TAPENADE. Indeed, when using TAPENADE, one should feed the set of subroutine codes to TAPENADE and specify the name of "top routine," as shown in Figure 29. In Appendix A, this procedure, accompanied by several simple examples, is presented.

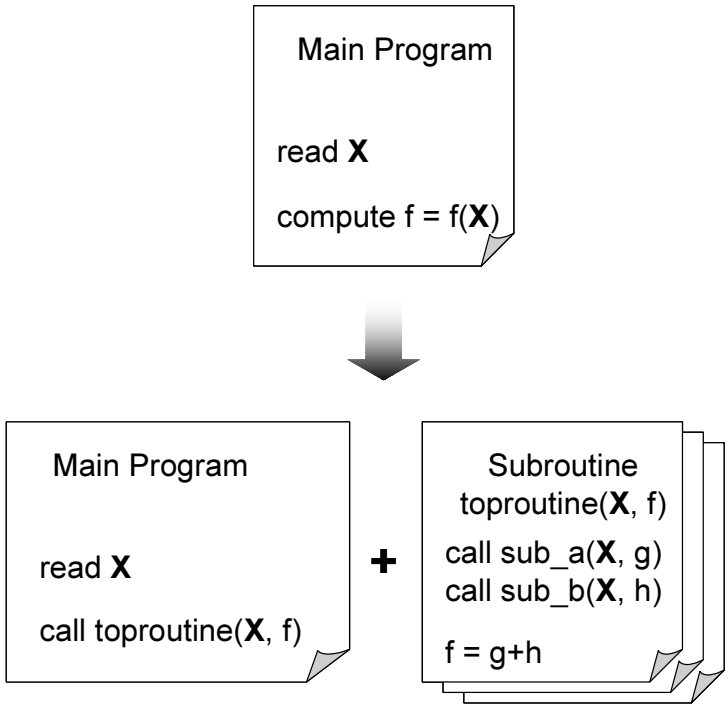


Figure 28: Form of a subroutines fed to TAPENADE.

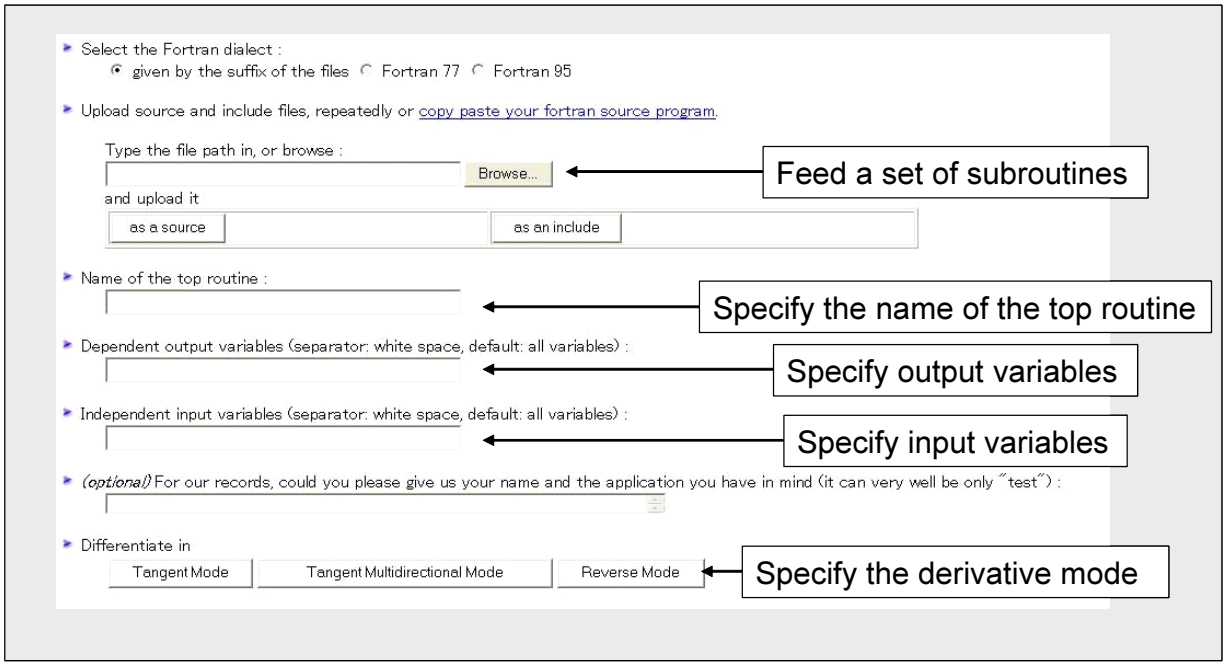


Figure 29: A screen shot of TAPENADE.

A summary of the three techniques for computing derivatives is given below in Table 7.

Table 7: Techniques for computing derivatives.

Method	Pros	Cons
FD Method	• Is easy to implement • Treats a computational analysis tool as a "black box" • Applies to any analysis tool	• Possesses an unknown size of appropriate Δh • Is vulnerable to noisy functions
Adjoint Method	• Computes derivatives analytically	• Requires a complicated preprocessing task and is thus difficult to implement
AD Method	• Computes derivatives analytically • Partially treats a computational analytical tool as a "black box" • Applies to any computational analysis tool if written in an appropriate language	• Requires computational analysis tools to be written in an appropriate language

Because the AD method is elegant as well as relatively easy to implement compared with the adjoint method, this study uses it to compute derivatives. However, since the general method for applying the AD method to CFD tools is not yet well known, it is the focus of this research and discussed in the following section.

4.2 Applying the AD Method to CFD

This section presents a method for calculating the derivatives of aerodynamics from existing codes through AD tools. In computing aerodynamics through the traditional CFD method, one must first generate a grid and then feed it to a flow solver that solves the system partial differential equation (PDE) iteratively, depicted in Figure 30. Likewise, computing aerodynamic derivatives also requires two stages: computing the derivatives of nodes with respect to parameters β, which control a configuration ($\frac{\partial \mathbf{x}}{\partial \beta}$) and computing the derivatives of aerodynamic loads with respect to nodes (e.g., $\frac{\partial C_l}{\partial \mathbf{x}}$, $\frac{\partial C_d}{\partial \mathbf{x}}$). Then, by multiplying both derivatives, one can obtain the derivatives of aerodynamic loads with respect to parameters β (e.g., $\frac{\partial C_l}{\partial \beta}$, $\frac{\partial C_d}{\partial \beta}$). The following subsections explain the methods for computing them.

4.2.1 Computing Derivatives of Nodes With Respect to Shape Parameters

Positions of nodes on a surface of an object can be represented by various curves: polynomial curves, Bezier curves, NURBS curves [64], and so on, which are generally controlled by shape parameters β. Therefore, nodes $\mathbf{x}$ that are on the surface and in space have some relationships with β, and one can compute $\frac{\partial \mathbf{x}}{\partial \beta}$ with the following methods:

Method 1: Using the FD method

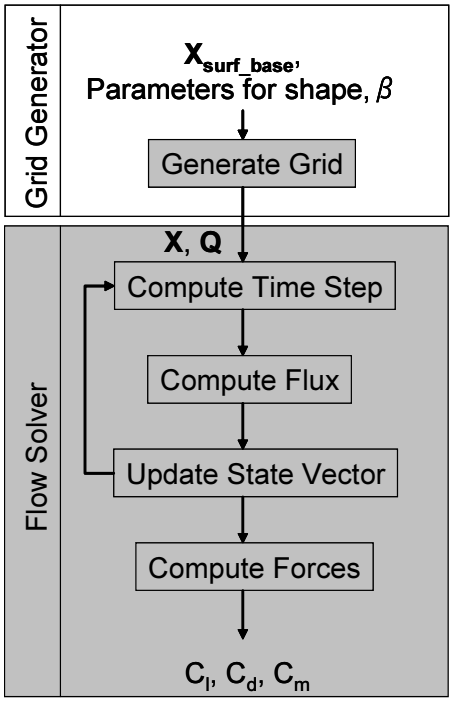


Figure 30: Flow chart of the traditional CFD method.

If a grid generator is a "black box" for users, but they can control the number of nodes, and the number of grid points of $\mathbf{x}(\beta)$ and that of $\mathbf{x}(\beta + \triangle\beta)$ are the same, computing $\frac{\partial \mathbf{x}}{\partial \beta}$ by means of the FD method is the easiest choice. When β is composed of n factors, mathematically, this method in forward difference is represented as follow:

$$\begin{pmatrix} \frac{\partial \mathbf{x}}{\partial \beta_1} \\ \vdots \\ \frac{\partial \mathbf{x}}{\partial \beta_i} \\ \vdots \\ \frac{\partial \mathbf{x}}{\partial \beta_n} \end{pmatrix} = \begin{pmatrix} \frac{\mathbf{x}(\beta+\Delta h \mathbf{a}_1)-\mathbf{x}(\beta)}{\Delta h} \\ \vdots \\ \frac{\mathbf{x}(\beta+\Delta h \mathbf{a}_i)-\mathbf{x}(\beta)}{\Delta h} \\ \vdots \\ \frac{\mathbf{x}(\beta+\Delta h \mathbf{a}_n)-\mathbf{x}(\beta)}{\Delta h} \end{pmatrix}, \tag{48}$$

where Δh is a small, user-dependent number, and $\mathbf{a}_i$ is a vector whose ith component is one and other components are all zeros. However, as explained in 4.1.1, although the FD method is the simplest, it is not recommended. Therefore, if other methods are available, it should not be used.

Method 2: Applying an AD tool directly to a grid generator

If one can read a program code of a grid generator written in appropriate languages, and the number of grid points of $\mathbf{x}(\beta)$ and that of $\mathbf{x}(\beta + \triangle\beta)$ are the same, $\frac{\partial \mathbf{x}}{\partial \beta}$ can be computed by a derivative code generated with an AD tool based on the program code of the grid generator. The structured grid generator may be categorized in this method. Since the number of grid points is much larger than the number of components of β, the FAD method should be used in the AD tool when the derivative code is generated.

Method 3: Introducing the spring-analogy method and applying an AD tool to the spring-analogy method

If one can not read a program code of a grid generator or if the number of grid points of $\mathbf{x}(\beta)$ and that of $\mathbf{x}(\beta + \triangle\beta)$ differ, the spring-analogy method [14] can be coupled with the grid generator, and $\frac{\partial \mathbf{x}}{\partial \beta}$ can be computed by a derivative code generated by an AD tool based on the code of the spring-analogy method. The unstructured grid [53][76] case may be categorized in this method because the number of nodes increases iteratively during their generation and because the numbers of nodes of $\mathbf{x}(\beta)$ and that of $\mathbf{x}(\beta + \triangle\beta)$ usually differ. In such a case, AD tools can not work properly for code for creating grid nodes even if it is written in an appropriate language for AD tools, probably because the code for creating grid nodes is not a series of functions differentiable with respect to the shape parameters β since the total number of nodes increases in each iteration. However, if the spring-analogy method is coupled with the grid generator, the numbers of nodes of $\mathbf{x}(\beta)$ and that of $\mathbf{x}(\beta + \triangle\beta)$ can be the same. Now, in the spring-analogy method, the following iterative steps are conducted:

Step 1: Compute the stiffness of all segments

The stiffness of a segment between two adjacent nodes i and j (Figure 31) is defined as an inverse of its length.

$$k_{i\,j} = \frac{1}{\sqrt{\left(x_j^0 - x_i^0\right)^2 + \left(y_j^0 - y_i^0\right)^2}}. \tag{49}$$

Here, the superscript "0" indicates that points are on the original locations. Therefore, the stiffness is computed before the deformation of nodes $\mathbf{x}$.

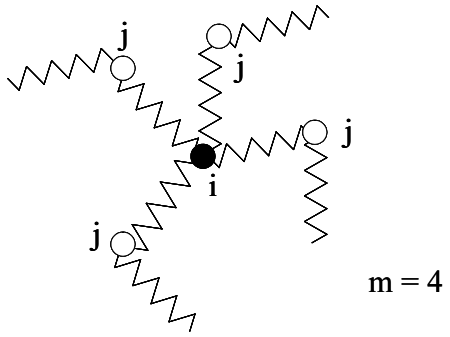


Figure 31: The spring analogy method.

Step 2: Deform the nodes on the surface

After the stiffness of all the segments is computed, only the nodes on the surface are forced to be deformed. Therefore, (x_i^0, y_i^0) becomes (x_i^1, y_i^1).

Step 3: Compute the new locations of the interior points iteratively

The new locations of the interior points are computed as follows:

$$x_i^{n+1} = x_i^{n-1} + \Delta x_i^n, \tag{50}$$

$$y_i^{n+1} = y_i^{n-1} + \Delta y_i^n, \tag{51}$$

where

$$\Delta x_i^n = \frac{\sum_{j=1}^{m} k_{ij} \delta x_{ij}^n}{\sum_{j=1}^{m} k_{ij}}, \tag{52}$$

$$\Delta y_i^n = \frac{\sum_{j=1}^{m} k_{ij} \delta y_{ij}^n}{\sum_{j=1}^{m} k_{ij}}, \tag{53}$$

$$\tag{54}$$

and where

$$\delta x_{ij}^n = \left(x_i^n - x_j^n\right) - \left(x_i^0 - x_j^0\right), \tag{55}$$

$$\delta y_{ij}^n = \left(y_i^n - y_j^n\right) - \left(y_i^0 - y_j^0\right). \tag{56}$$

In Equations 50 to 56, n starts from one. In Equations 52 and 53, m is the number of nodes that are adjacent to node i. Step 3 is repeated until the locations of (x_i, y_i) do not change within some tolerance.

Figure 32 shows an example of a two-dimensional unstructured grid deformed by the spring-analogy method.

If the deformation is large, this original spring-analogy method is reported to have problems; thus, many other improved spring-analogy methods have been developed [17], [56]. However, computing the derivatives is the main concern at this stage (i.e., deformation can be assumed to be very small), so the original spring-analogy method can safely be used.

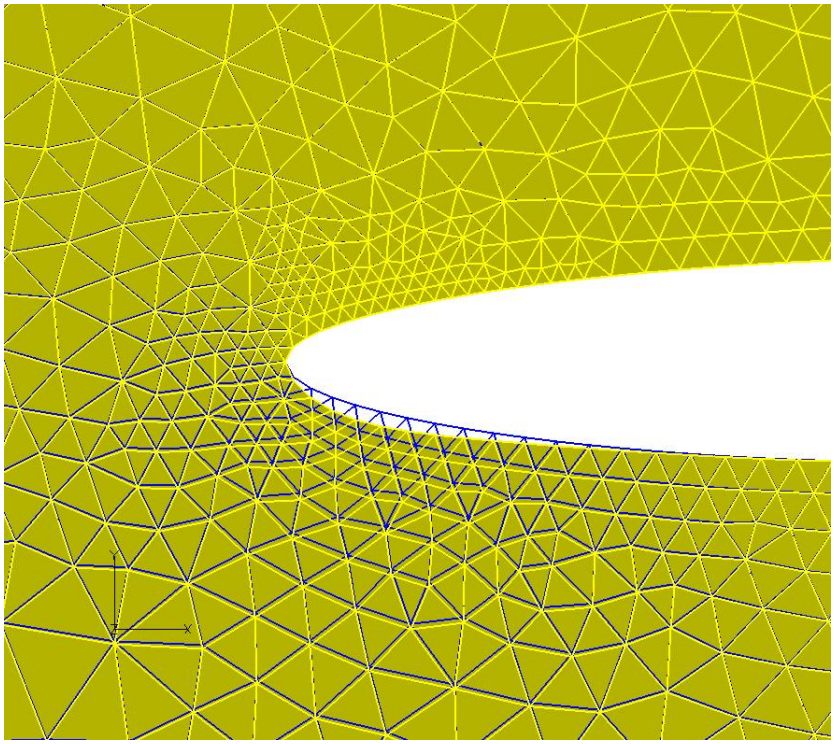


Figure 32: A two-dimensional unstructured grid deformed by the spring-analogy method (blue: before deformation, yellow: after deformation).

4.2.2 Computing the Derivatives of Aerodynamics With Respect to Nodes

In this stage, the derivatives of aerodynamic coefficients with respect to the locations of nodes (e.g., $\frac{\partial C_l}{\partial \mathbf{x}}$, $\frac{\partial C_d}{\partial \mathbf{x}}$) are computed. Therefore, a CFD code is applied to an AD tool; the locations of nodes $\mathbf{x}$ are specified as input variables, and the aerodynamic coefficients (e.g., C_l, C_d) are specified as output variables in an AD tool in order to obtain a derivative code. Here, if the FAD mode is selected, $\frac{\partial \mathbf{x}}{\partial \beta}$ is first assigned to the generated derivative code, and $\frac{\partial C_l}{\partial \beta}$ or $\frac{\partial C_d}{\partial \beta}$ is computed for each factor of β. However, if the RAD mode is selected, the generated derivative code computes $\frac{\partial C_l}{\partial \mathbf{x}}$ or $\frac{\partial C_l}{\partial \mathbf{x}}$ first, and one should multiply $\frac{\partial \mathbf{x}}{\partial \beta}$ next in order to compute $\frac{\partial C_l}{\partial \beta}$ or $\frac{\partial C_d}{\partial \beta}$. Therefore, if the number of factors of shape parameter β is greater than that of aerodynamics, which mostly occur in real problems, the RAD mode is selected; otherwise, the FAD mode is selected. The two modes for computing the derivatives of aerodynamics with respect

to nodes are depicted in Figure 33.

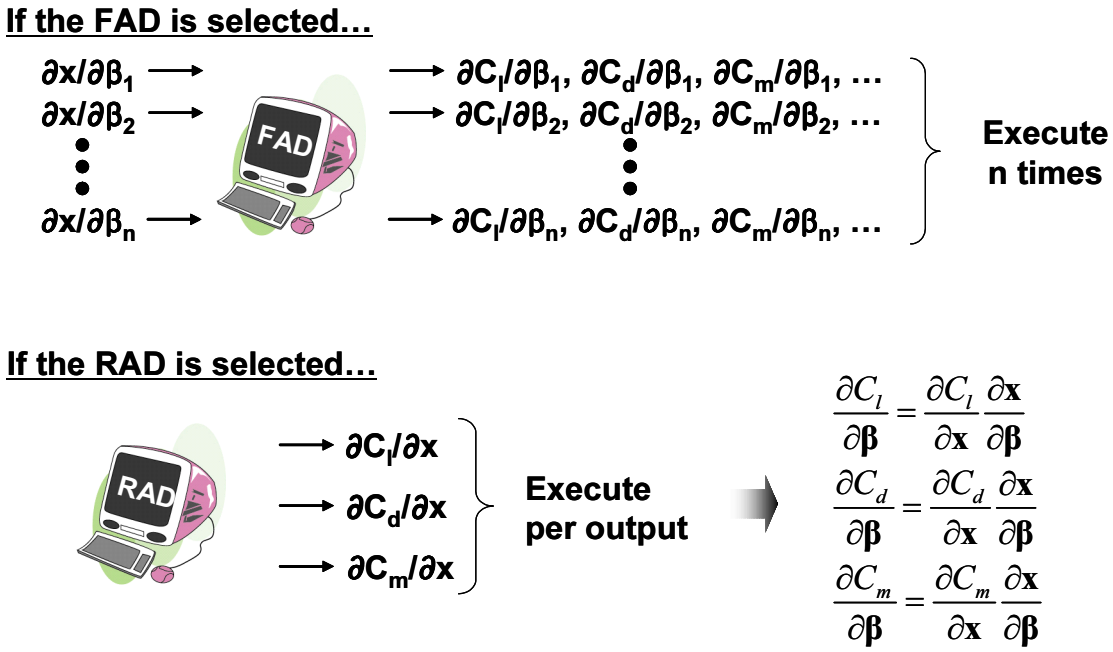


Figure 33: The two modes for computing the derivatives of aerodynamics with respect to nodes.

After an adjoint code using the AD tool is obtained, it may have the structure depicted in Figure 34. Here, the loops for solving the system PDE are intentionally opened so that readers can understand that this adjoint code has many steps to compute, and dotted-line and solid-line boxes have the same meaning as in Figure 26. As explained in 4.1.3.2, some intermediate results in the dotted-line box should be stored in memory and recalled during the path through the solid-line box.

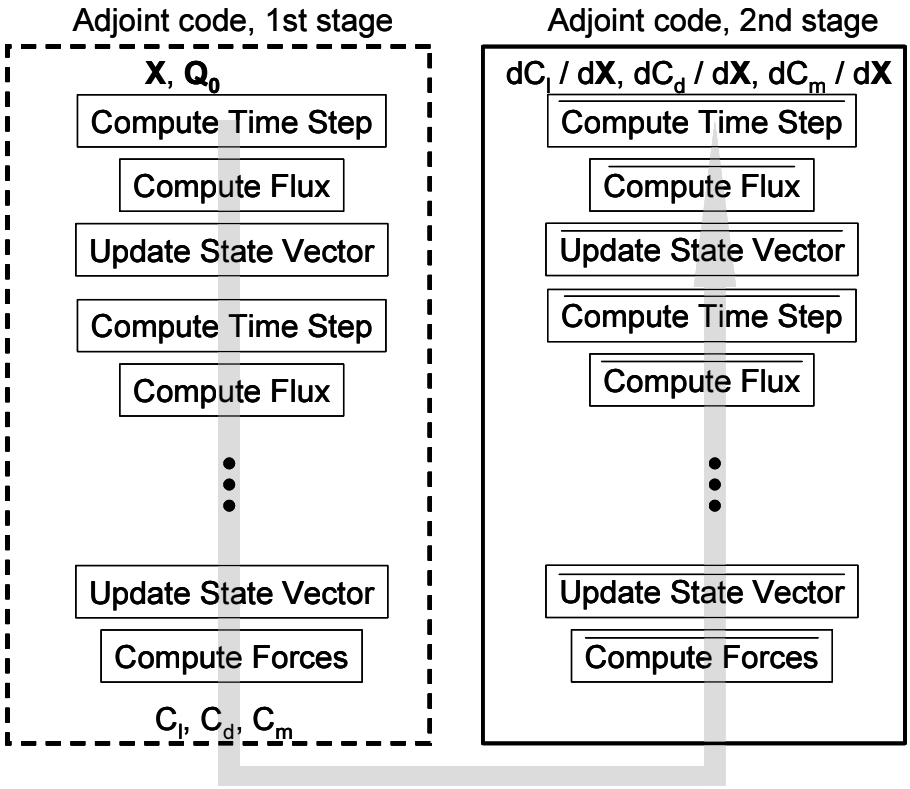


Figure 34: A flow chart of an adjoint code of a flow solver generated by an AD tool.

However, directly using this adjoint code is highly inefficient because of the tremendous number of intermediate results that need to be solved during the iteration in the first stage of the adjoint code, which could easily exceed the memory capacity of the hardware. Therefore, a converged state vector is computed beforehand using the CFD solver and fed to the adjoint code so that memory capacity is not exceeded. By doing so, the iteration in the first stage of the adjoint code can be omitted because the state vector has already converged and because one can assume that a set of intermediate results computed in a loop is the same as those in the next loop. This strategy saves considerable memory and computation time. Figure 35 illustrates this strategy.

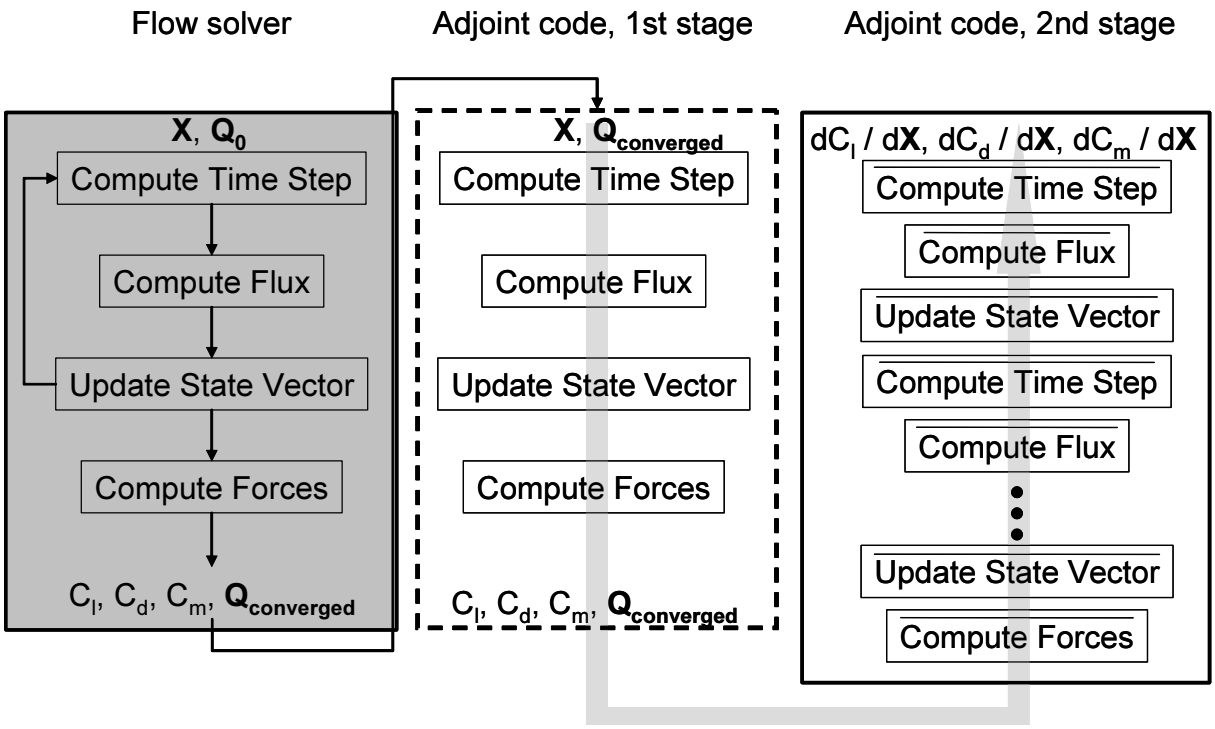


Figure 35: The flow chart of the proposed strategy for computing the derivatives of aerodynamics with respect to nodes.

4.2.3 Summary of Applying the AD Tool to CFD and Its Validation

Figure 36 shows the general procedure for computing aerodynamic coefficients and their derivatives. The procedure is divided into five stages, A through E. In each stage, computer software is shown inside a box, and its inputs are located above the software, and outputs are located below the software. Between the different stages, inputs and outputs are transferred, and in such a case, they are connected by arrows in Figure 36. For each stage, input and output are explained as follows:

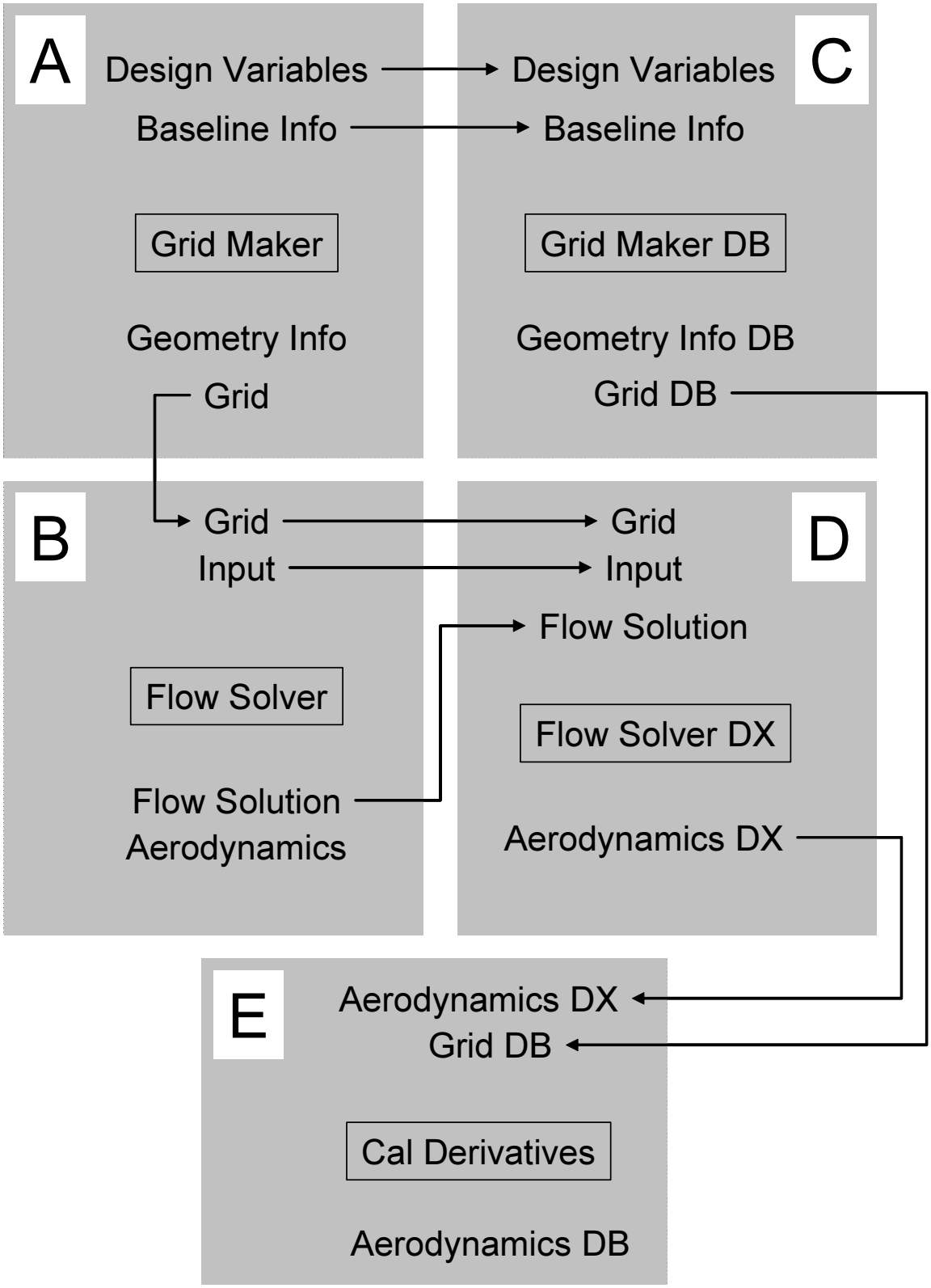


Figure 36: A general procedure for computing aerodynamics and the derivatives of aerodynamics.

Stage A

"Design Variables" are shape parameters β.

"Baseline Information" is x, y (two-dimensional) or x, y, z (three-dimensional) coordinates of an original surface geometry.

"Geometry Info" may include the area or the volume of a generated geometry.

"Grid" contains grid information about a generated geometry readable for a flow solver.

Stage B

"Input" includes all information necessary to compute aerodynamic coefficients with a CFD solver (e.g., CFL number).

"Flow Solution" is a converged state vector $\mathbf{Q}_{converged}$.

"Aerodynamics" may include C_l, C_d or C_L, C_D.

Stage C

"Grid DB" contains information about $\frac{\partial \mathbf{x}}{\partial \beta}$.

"Geometry Info DB" may contain the $\frac{\partial (area)}{\partial \beta}$ or $\frac{\partial (volume)}{\partial \beta}$ of a generated geometry.

"DB," as in "Grid Maker DB," "Grid DB," and "Geometry Info DB," represents the derivatives with respect to shape parameters β.

Stage D

"Aerodynamics DX" may contain information about $\frac{\partial C_l}{\partial \mathbf{x}}$, $\frac{\partial C_d}{\partial \mathbf{x}}$ or $\frac{\partial C_L}{\partial \mathbf{x}}$, $\frac{\partial C_D}{\partial \mathbf{x}}$.

"DX," as in "Flow Solver DX" and "Aerodynamics DX," represents derivatives with respect to nodes $\mathbf{x}$.

Stage E

"Aerodynamics D" may contain information about $\frac{\partial C_l}{\partial \beta}$, $\frac{\partial C_d}{\partial \beta}$ or $\frac{\partial C_L}{\partial \beta}$, $\frac{\partial C_D}{\partial \beta}$. It is computed by multiplying "Grid D" and "Aerodynamics DX."
"DX," as in "Aerodynamics DX," represents the derivatives with respect to nodes $\mathbf{x}$.
"DB," as in "Grid DB" and "Aerodynamics DB," represents the derivatives with respect to shape parameters β.

As validation of the proposed method, $\frac{\partial C_l}{\partial \alpha}$ of NACA 0012 [5], whose flight Mach number is 0.75, is computed by three different methods: the FD, the FAD, and the RAD. The governing equation is the full potential equation (FPE) [9][47], and a structured o-grid, shown in Figure 37, is used. The FPE solver and the o-grid generator are coded by Sankar [73] in Fortran 77.

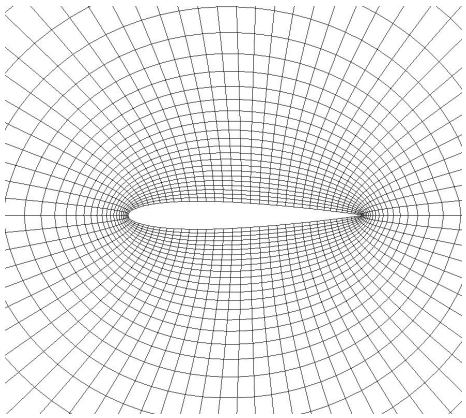


Figure 37: The structured o-grid around NACA 0012.

Figures 38 and 39 show C_l trends between the angle of attack α 1.0[deg] to 3.0 [deg] and 1.98 [deg] to 2.12 [deg], respectively. Figure 38 shows a non-linear effect at the angle of attack greater than $\alpha = 2.0$ [deg], which is probably due to the shock wave. Table 8 summarizes $\frac{\partial C_l}{\partial \alpha}$s at the angle of attach $\alpha = 2.0$ [deg], computed by the FD with several α increments, FAD, and RAD, and it shows all these three methods

compute similar values. This study shows that the proposed method accurately computes derivatives.

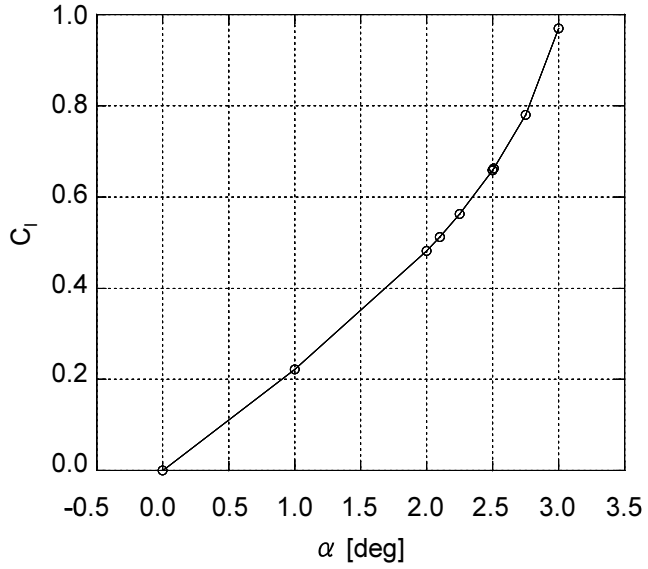


Figure 38: C_l vs. α of NACA 0012 at $M_\infty = 0.75$ ($1.0 \leq \alpha \leq 3.0$).

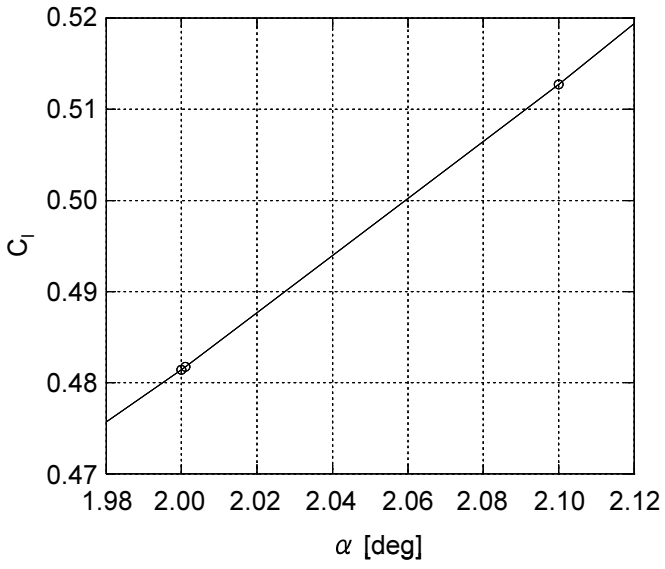


Figure 39: C_l vs. α of NACA 0012 at $M_\infty = 0.75$ ($1.98 \leq \alpha \leq 2.12$).

Table 8: $\frac{\partial C_l}{\partial \alpha}$ at $\alpha = 2.0$[deg] computed by the FD, the FAD, and the RAD.

	FD	FAD	RAD
$\frac{\partial C_l}{\partial \alpha}$	$0.3060(\Delta\alpha = 10^{-6})$	0.2991	0.2991
	$0.3052(\Delta\alpha = 10^{-3})$		
	$0.3127(\Delta\alpha = 10^{-1})$		

4.3 Modifying the Governing Equation of the Trust Region Ratio

In 3.2.2, the author found that the original governing equation of the trust region ratio in Equations 23 to 27 sometimes does not work properly due to its strictness against violations of constraints. Thereby, a new governing equation of the trust region ratio, which is more tolerant against violations of constraints, is proposed.

Equation 57 shows the proposed governing equation of the trust region ratio. For the purpose of explanation, each term in Equation 57 is labeled N_1, D_1, and so forth, in Equation 58. For each term, the reasons for switching from the original to new term are described below:

$$\rho_n = \frac{f_{high}(\mathbf{x}_n) - \left\{ f_{high}(\mathbf{x}_c) + r_p \sum_{i=1}^{m} \max\left[0, g_{high,i}(\mathbf{x}_c) - tolg_i\right] \right\}}{f_{surrogate}(\mathbf{x}_n) - f_{surrogate}(\mathbf{x}_c)} \tag{57}$$

$$= \frac{N_1 - N_2}{D_1 - D_2}. \tag{58}$$

The reason for switching from the original to new term in N_2

Only N_2 deals with the violation of constraints, and a tolerance, or a user-dependent or problem-dependent vector $tolg_i$, is newly introduced because surrogate constraints

$g_{surrogate,i}(\mathbf{x}_c) = 0$ may be not sufficiently accurate to mimic $g_{high,i}(\mathbf{x}_c) = 0.0$, especially when the base point $\mathbf{x}_n$ is located far from points that satisfy $g_{high,i}(\mathbf{x}) = 0.0$, where an optimum design point is located in most optimization cases.

The reason for switching from the original to new term in D_2

Even if $\mathbf{x}_c$ violates the surrogate constraints, $\mathbf{x}_c$ should be accepted if $f(\mathbf{x}_c)$ satisfies the true constraints. Therefore, D_2 takes the form of a normal function, not one of the penalty function.

The reason for switching from the original to new term in N_1

Once $\mathbf{x}_c$ is accepted, it will be $\mathbf{x}_n$ in the next iteration. Since the violation of the constraints is taken care of in N_2, N_1 takes the form of a normal function, not one of the penalty function.

The reason for switching from the original to new term in D_1

Since, by definition, D_1 and N_1 should be the same, D_1 takes the form of a normal function, not one of the penalty function.

The proposed governing equation of the trust region ratio can lead to a new design point near the constraint with high fidelity ($g_{high} = 0$), even if this design point violates it, and one can expect the AMF to create better surrogate functions near constraints from the next iteration. This mechanism is depicted in Figure 40.

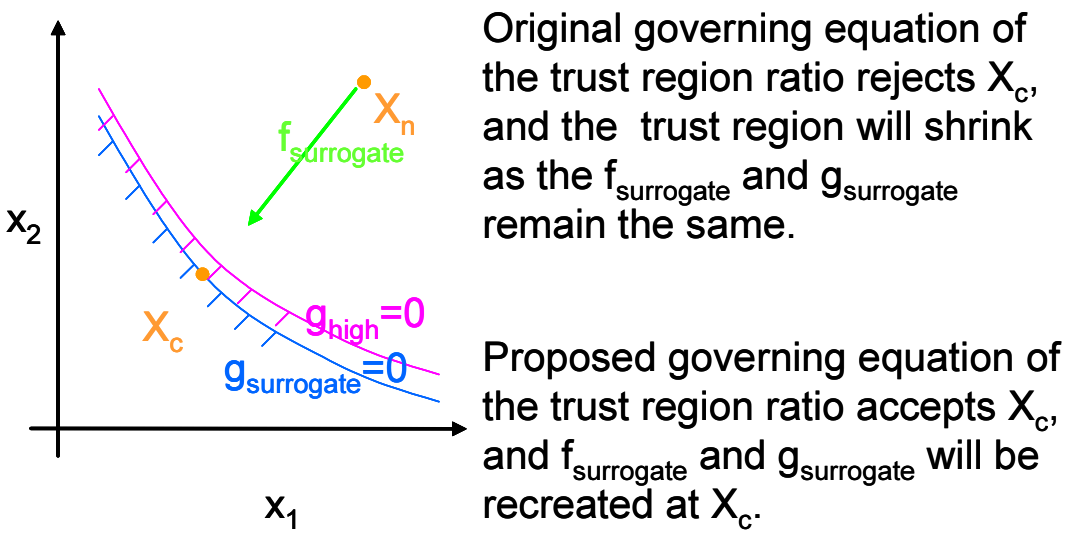


Figure 40: The differences between the original governing equations of the trust region ratio and the proposed governing equations of the trust region ratio.

Now, let us refer to the AMF with the proposed governing equation of the trust region ratio as the "modified AMF." As a verification of the modified AMF, a small analytical constrained problem, solved in 3.2.2 by the AMF with the original governing equation of the trust region ratio, is solved again by the modified AMF. To ensure a fair comparison of the results, we use the same criteria and initial trust region size (i.e., $\varepsilon_f = \varepsilon_{\mathbf{x}} = 1.0e-4$, and the initial trust region size is 100% of the entire domain). The newly-introduced term $tolg_i$ is 10^{-2}. Figure 41 shows relationships between the initial points and the corresponding converged solutions. Figures 42 and 43 show the relationships between the initial points and the corresponding numbers of the high- and low-fidelity function calls, respectively.

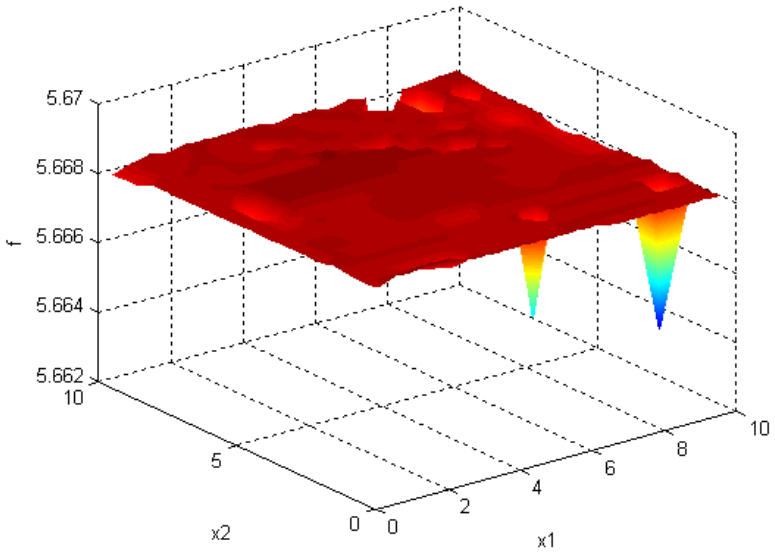


Figure 41: The relationships between the initial points and the corresponding converged solutions (Modified AMF).

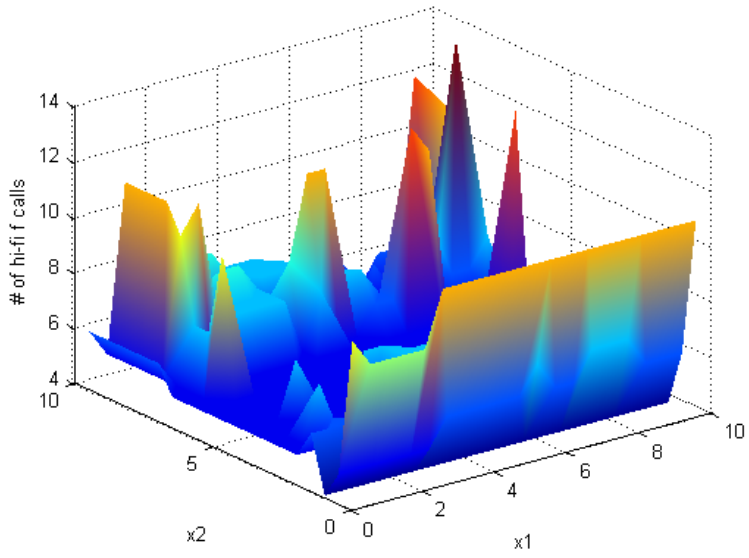


Figure 42: The relationships between the initial points and the corresponding numbers of the high-fidelity function calls (Modified AMF).

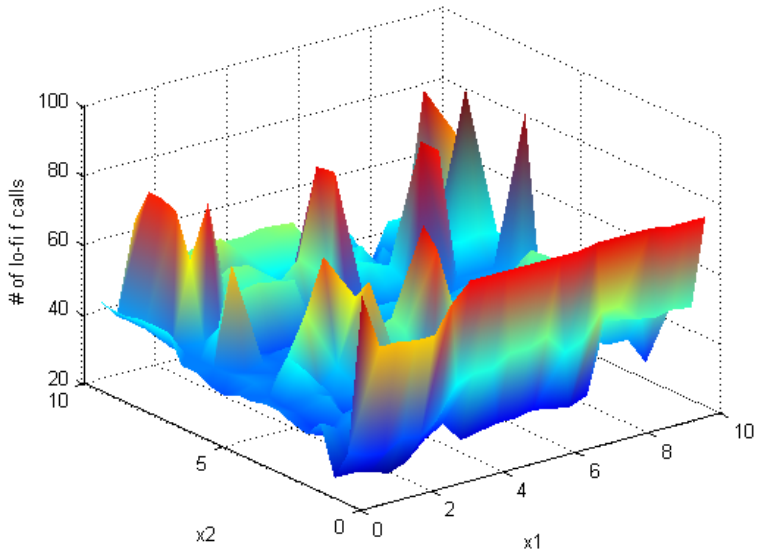


Figure 43: The relationships between the initial points and the corresponding numbers of the low-fidelity function calls (Modified AMF).

From Figure 41, one can conclude that the modified AMF yields more correct converged results regardless of the initial points; and Figures 42 and 43 show that these results are obtained with far fewer iterations than those of the original AMF.

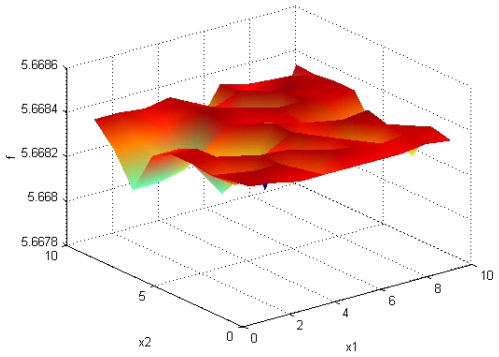
Solving this analytical constrained problem with (8.0, 1.0) as the initial point is interesting because the original AMF cannot find a correct answer when (8.0, 1.0) is an initial point. Table 9 summarizes the results, and as references, the results from the original AMF and the SQP, which are already presented in Table 4, are included in Table 9.

Table 9: Results for the constrained problem with the initial point at $(8.0, 1.0)$.

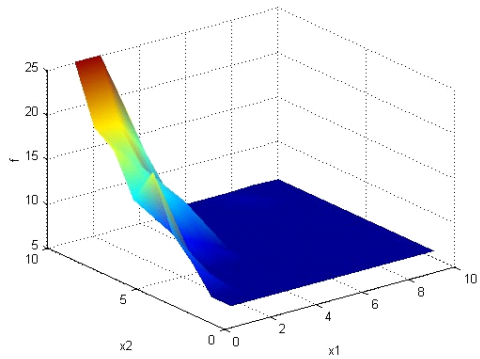
	Modified AMF $\varepsilon_f = \varepsilon_{\mathbf{x}}$ =1.0e-4	Modified AMF $\varepsilon_f = \varepsilon_{\mathbf{x}}$ =1.0e-5	Original AMF $\varepsilon_f = \varepsilon_{\mathbf{x}}$ =1.0e-4	SQP $\varepsilon_f = \varepsilon_{\mathbf{x}}$ =1.0e-4
# of hi-fi function calls	4	10	100	22
# of lo-fi function calls	25	58	300	-
Final hi-fi function	5.6677	5.6683	222.4729	5.6684
Final x_1	0.8833	0.8842	7.3927	0.8842
Final x_2	1.1521	1.1507	0.5363	1.1507

From Table 9, one can conclude that the modified AMF finds a much better answer than the original AMF. However, the value of the converged objective function by the modified AMF with $\varepsilon_f = \varepsilon_{\mathbf{x}} = 1.0e{-}4$ slightly differs from that by the SQP. Therefore, the modified AMF with more strict convergence criteria (i.e., $\varepsilon_f = \varepsilon_{\mathbf{x}} = 1.0e-5$) is conducted. The results are also presented in Table 9. In this case, the modified AMF finds almost the same answer as the SQP, but the number of high-fidelity function calls is still less than half of the number of the SQP, indicating that the modified AMF works better than the SQP.

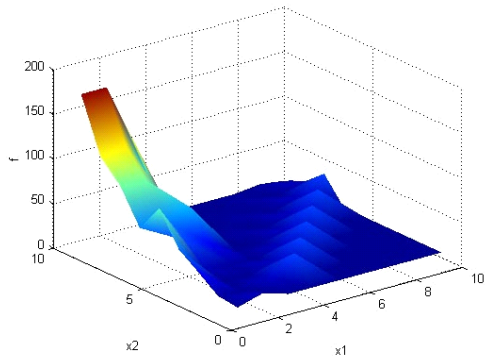
Finally, the effect of the size of *tolg* in the small analytical constrained problem is investigated. Figure 44 shows the relationships between the initial points and the corresponding final solutions with different sizes of *tolg*s.



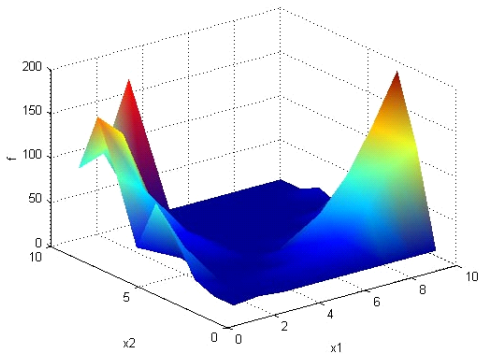
(a) $tolg = 10^{-4}$



(b) $tolg = 10^{-6}$



(c) $tolg = 10^{-8}$



(d) $tolg = 0.0$ (Original AMF)

Figure 44: The effect of the size of $tolg$.

From Figure 44, one can observe that the larger size of $tolg$ leads to a wider search range in the optimization because final solutions are closer to the true optimum value wherever the initial point is. In addition, one should note that increasing the size of $tolg$ is not identical to relaxing the constraints because final solutions converge to the true optimum point. However, since the AMF with the new governing equation of the trust region ratio may end up the optimization in the infeasible region within the tolerance of $tolg$, one should be careful in choosing the appropriate size of $tolg$.

4.4 Robust AMF

The previous sections have proposed innovative ideas that enhance the robustness of the original AMF. With the AD method, the AMF can perform better under a noisy environment, and the computational time required does not depend on the number of design variables. With the modified governing equation of the trust region ratio, the AMF is more likely to find the correct answer in optimization problems under constraints. Hence, this study has incorporated these ideas into the AMF to create the "Robust AMF" (Figure 45). In the following chapter, the Robust AMF is applied to several aerospace engineering problems and its effectiveness is validated.

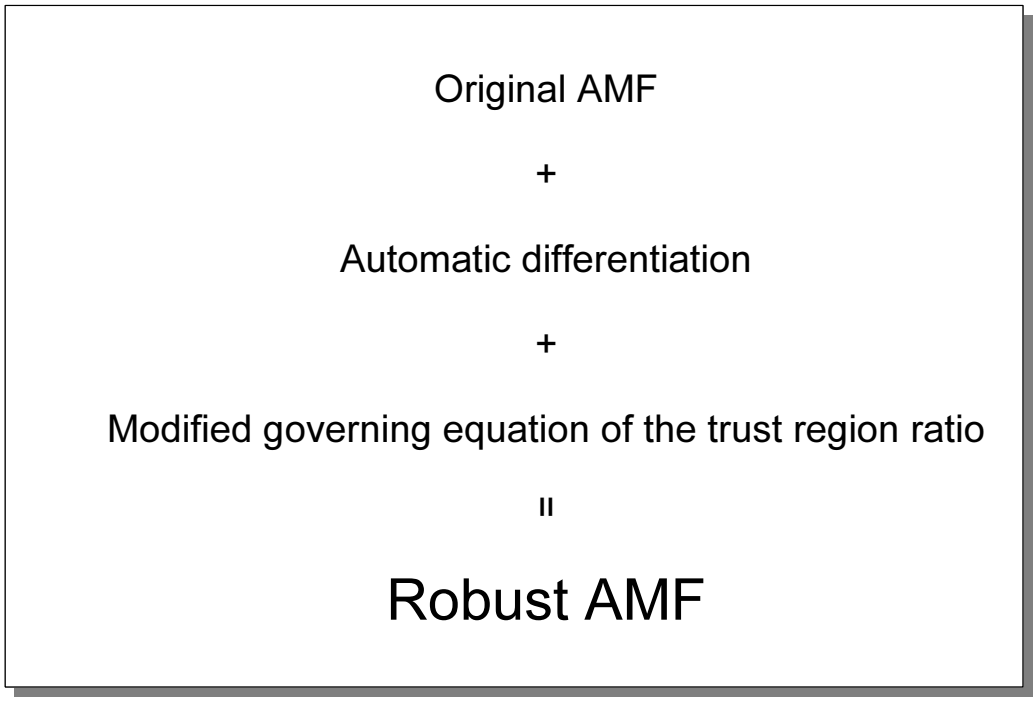


Figure 45: Robust AMF.

CHAPTER V

DEMONSTRATIONS OF THE ROBUST AMF FOR AEROSPACE ENGINEERING PROBLEMS

In this chapter, optimization problems in aerospace engineering are solved with the Robust AMF, the original AMF, and the SQP. In all methods, derivatives are computed through the derivative code generated by the AD tool. For each problem, the involved software is first validated, and then the different optimization methods are compared. Through these problems, the effectiveness of the Robust AMF is discussed.

5.1 Design of an Airfoil in the Transonic Speed Regime

The first optimization problem, the design of an airfoil flying at $M_\infty = 0.8$ with an angle of attack $\alpha = 0.0$ [deg], is defined as follows:

$$\underset{\beta}{\text{Minimize}}: \quad C_d, \tag{59}$$

$$\text{Subject to}: \quad C_l > C_{l,initial}, \tag{60}$$

$$S > S_{initial}, \tag{61}$$

where β is a vector that controls the shape of the airfoil, and S is the non-dimensional enclosed area of the airfoil when the chord length is set to one. As initial airfoils, RAE 2822 [83] RAE 5212, and RAE 5214 are selected. While all details are discussed in RAE 2822 case, only the final results are discussed in RAE 5212 and RAE 5214 cases. However, in the problem setting, everything except the initial geometry is same between each case.

The flow solver used in this problem is a Navier-Stokes/Euler solver NSC2KE [82], and the mesh generator is an unstructured mesh generator BAMG [42]. Basic information about these programs are summarized in Table 10.

Table 10: Mesh generator and flow solver used in airfoil optimizations.

Mesh generator	Flow solver
BAMG [42]	NSC2KE [82]
• Two-dimensional unstructured triangular mesh • Solution-based adaptation [63]	• Finite volume Galerkin code • Explicit fourth-order Runge-Kutta scheme [29] for time integration • Roe scheme [69] with limited second order for flux evaluation • Two-dimensional Euler or two-dimensional Navier-Stokes with the $\overline{\kappa} - \varepsilon$ turbulence model [49]

Because computations with the Navier-Stokes equation require more computational power and because computer resources are limited, this study employed the Euler mode in NSC2KE. For the low-fidelity model, a coarse unstructured mesh shown in Figure 46 is used, and for the high-fidelity model, a solution adapted unstructured mesh shown in Figure 47 is used. The coarse unstructured mesh has approximately 1,000 to 2,000 nodes while the solution adapted unstructured mesh, which is generated after five solution adaptations, has approximately 2,500 to 3,500 nodes. As shown in Figures 46 and 47, the high-fidelity model has superior mesh resolutions around both the leading edge of the airfoil and the shock waves on the upper and lower surfaces of the airfoil, which may lead to considerable variation between the computed aerodynamics of the high- and low-fidelity models.

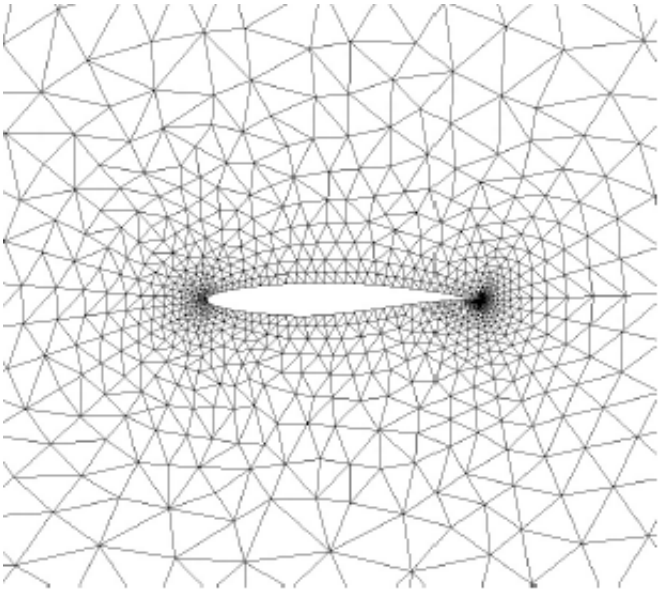


Figure 46: Low-fidelity model (1,000-2,000 nodes).

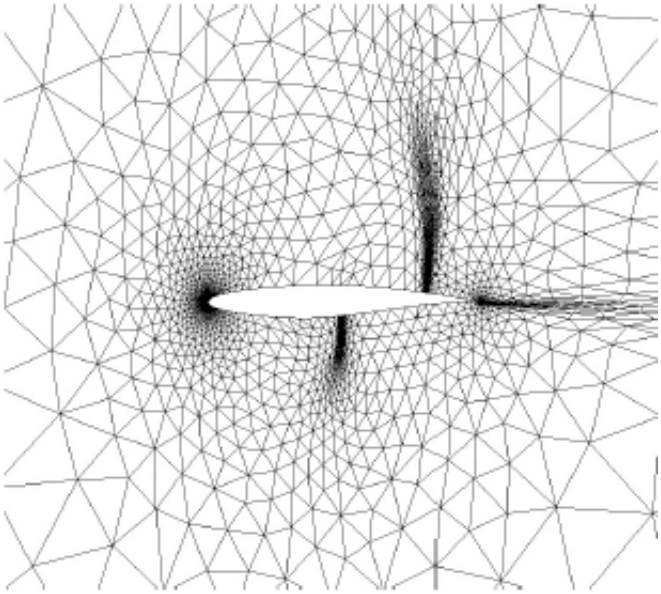


Figure 47: High-fidelity model (2,500-3,500 nodes).

In order to change the airfoil shape in the optimizations, this study uses the Hicks-Henne method [43], described in Equations 62 and 63. This method is able to generate

a variety of airfoils with a limited number of design variables [67]. Curves, referred to as "Hicks-Henne functions," shown in Figure 48, are linearly added to the base airfoil shape with some weight β_is. The solid curves shown in Figure 48 are for changing the curvatures of both the upper and lower airfoil surfaces, and one dotted curve shown in Figure 48 is for changing the location of the airfoil trailing edge. Therefore, a total of 21 $(= 2 \times 10 + 1$) design variables $(\beta_i, i = 1, ..., 21)$ control the shape of the airfoil in this study.

$$\begin{aligned} y_{new-lower} = y_{base-lower} & \\ & - \beta_1 \frac{10\,(1-x)\,x}{\exp(20x)} \\ & - \beta_2 \frac{10\,(1-x)\,x}{\exp(40x)} \\ & - \beta_3 \frac{\sqrt{x}\,(1-x)}{\exp(3x)} \\ & - \sum_{i=4}^{10} \beta_i \sin^5\left(\pi x^{b_i}\right) + \beta_{21} x^{10}, \end{aligned} \tag{62}$$

$$\begin{aligned} y_{new-upper} = y_{base-upper} & \\ & + \beta_{11} \frac{10\,(1-x)\,x}{\exp(20x)} \\ & + \beta_{12} \frac{10\,(1-x)\,x}{\exp(40x)} \\ & + \beta_{13} \frac{\sqrt{x}\,(1-x)}{\exp(3x)} \\ & + \sum_{i=14}^{20} \beta_i \sin^5\left(\pi x^{b_i}\right) + \beta_{21} x^{10}, \end{aligned} \tag{63}$$

where

$$
\begin{aligned}
b_{4,14} &= 0.5757166, \\
b_{5,15} &= 0.7564708, \\
b_{6,16} &= 1.0, \\
b_{7,17} &= 1.356915, \\
b_{8,18} &= 1.943358, \\
b_{9,19} &= 3.106283, \\
b_{10,20} &= 6.578813.
\end{aligned}
$$

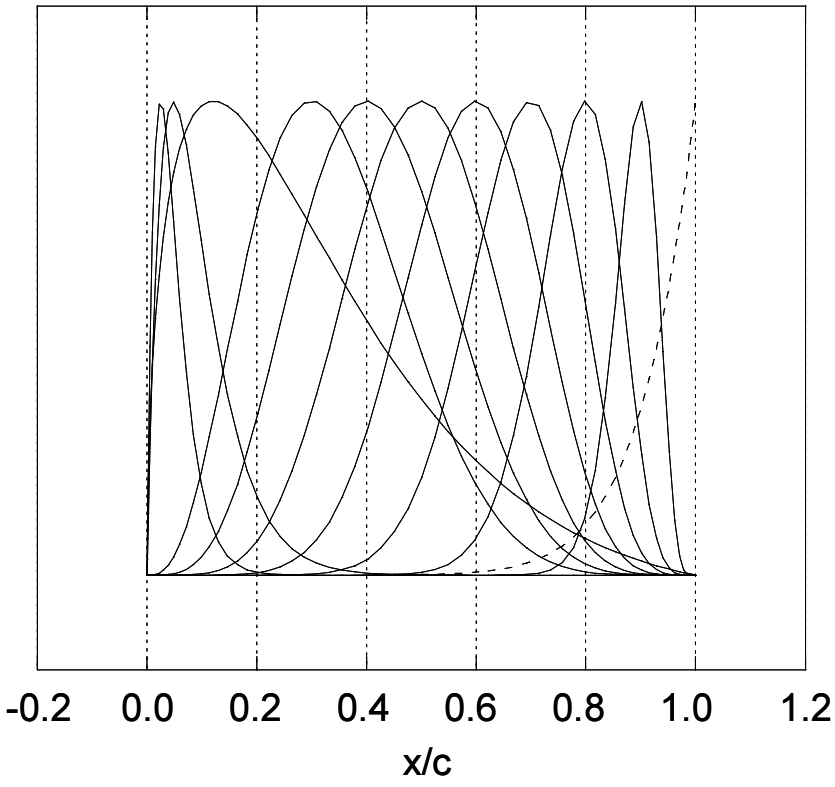


Figure 48: Hicks-Henne functions.

5.1.1 Validations of the Software

Validations for the flow solver and the derivative solver are conducted below. At the end of this subsection, their required CPU times and memory sizes are summarized in Table 13.

5.1.1.1 The flow solver

At Mach 0.75 with a 3.0 [deg] angle of attack, a numerical simulation of RAE 2822, conducted by AGARD, which may be considered as a standard reference for CFD analysis in an inviscid case (Euler), is available [83]. Therefore, although numerical accuracy is not the main focus of this study, validation of the software is done by comparing the results of the software with the AGARD results. Figure 49 presents a comparison of pressure coefficients around RAE 2822, Figure 50 presents a comparison of C_l, and Figure 51 presents a comparison of C_d, computed with the low- and high-fidelity models in this study, and by AGARD. The AGARD results show lower pressure on the upper surface and higher pressure on the lower surface than others in Figure 49. Therefore, AGARD result shows higher C_l in Figure 50. However, because numerical accuracy is not the main focus of this study and because the pressure coefficient of this flight condition is very sensitive to all related variables, this result is accepted as is. More important are the different tendencies of the high- and low-fidelity models. In the high-fidelity model, shock is captured very sharply, as in the AGARD case, while it is smeared in the low-fidelity model. In addition, since the mesh resolution of the high-fidelity model is higher than that of the low-fidelity model around both the leading edge and the shock wave, as seen in Figures 46 and 47, variations in the pressure distributions of both models may occurs when different airfoil shapes or flight conditions are considered. However, Figure 49 shows that they are not as discrepant in the RAE 2822 case.

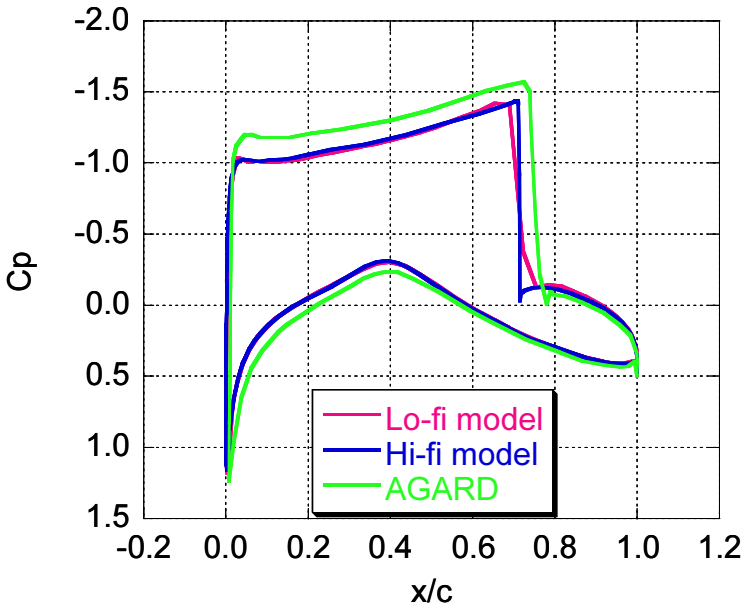


Figure 49: Comparison of the pressure coefficients around RAE 2822 ($M_\infty = 0.75$, $\alpha = 3.0$ [deg]).

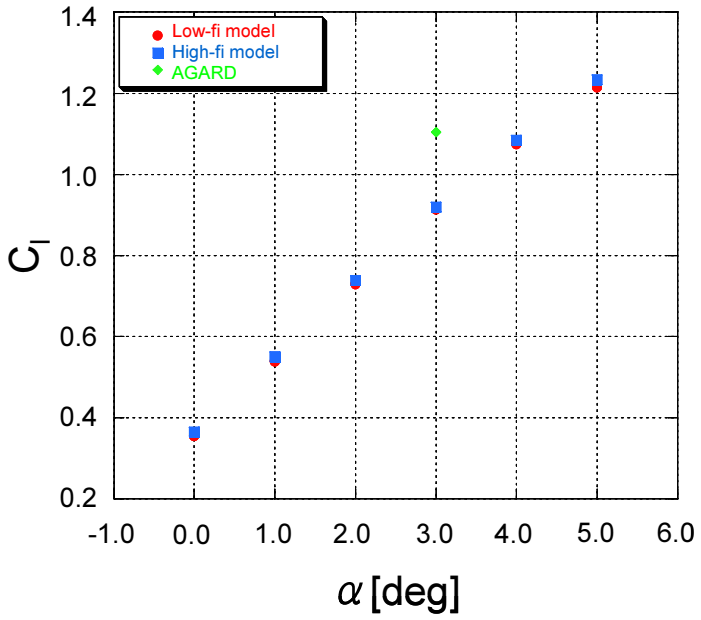


Figure 50: Comparison of the lift coefficients of RAE 2822 ($M_\infty = 0.75$).

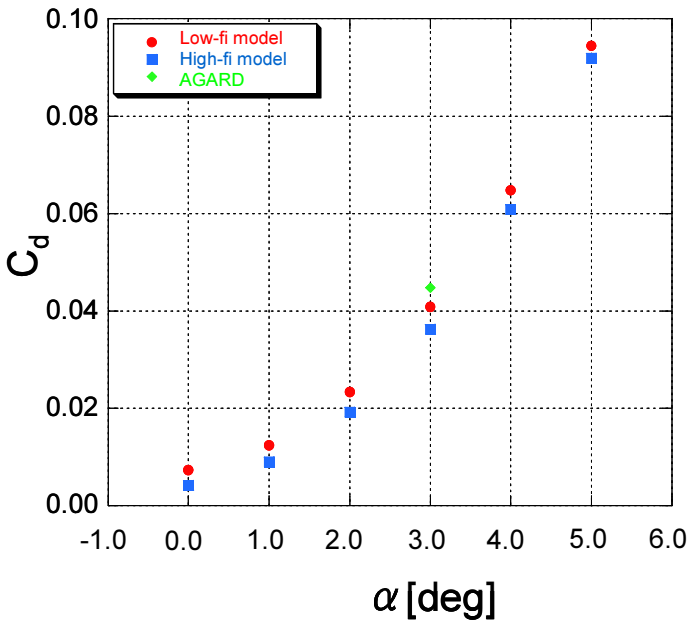


Figure 51: Comparison of the drag coefficients of RAE 2822 ($M_\infty = 0.75$).

In this study, optimizations are implemented at $M_\infty = 0.8$ with an angle of attack of $\alpha = 0.0$ [deg]. Under such flight conditions, typical convergence histories about a normalized residual with the low- and high-fidelity models are shown in Figures 52 and 53, respectively. In both cases, this study uses the local time stepping algorithm [52] to obtain converged solutions in a shorter time and CFL numbers of 1.5, which produces stable convergence. Judging from Figures 52 and 53, 1,000 iterations are probably sufficient for the low-fidelity case, and 2,500 for the high-fidelity case in the entire optimization process because after these numbers of iterations, normalized residuals are less than 10^{-3}, one of the criteria for convergence.

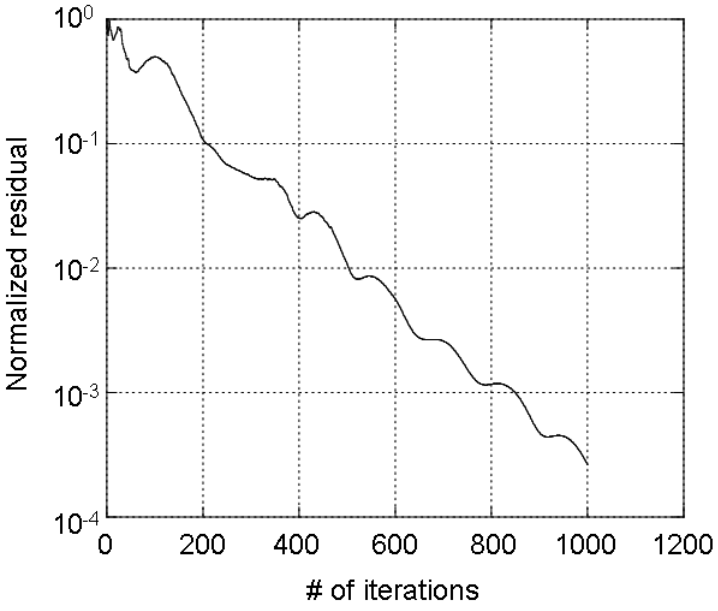


Figure 52: Convergence history of a normalized residual with the low-fidelity model (RAE 2822, M=0.8, AoA=0.0 [deg]).

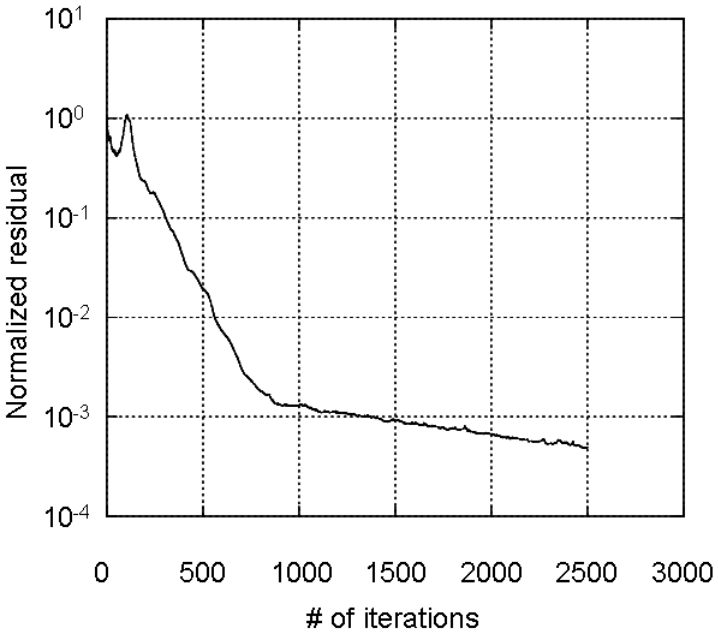


Figure 53: Convergence history of a normalized residual with the high-fidelity model (RAE 2822, M=0.8, AoA=0.0 [deg]).

Additionally, histories of C_l and C_d with both the low- and high-fidelity models are presented in Figures 54, 55, 56, and 57. These figures also justify the number of iterations set for the both models.

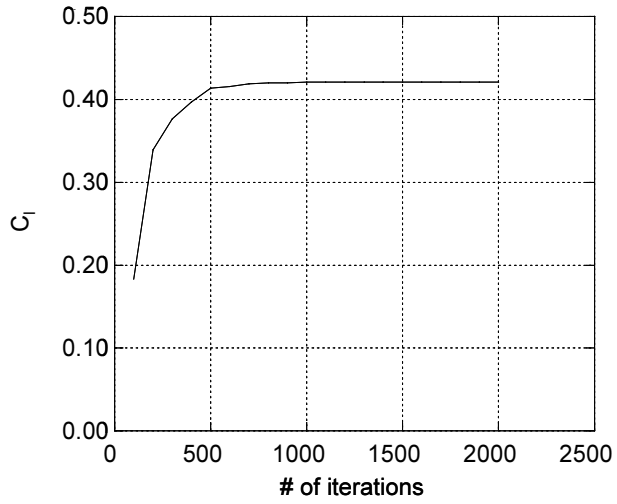


Figure 54: Convergence history of C_l with the low-fidelity model (RAE 2822, M=0.8, AoA=0.0 [deg]).

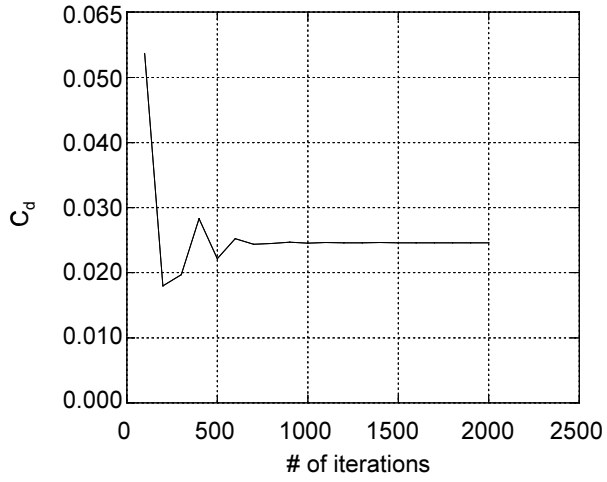


Figure 55: Convergence history of C_d with the low-fidelity model (RAE 2822, M=0.8, AoA=0.0 [deg]).

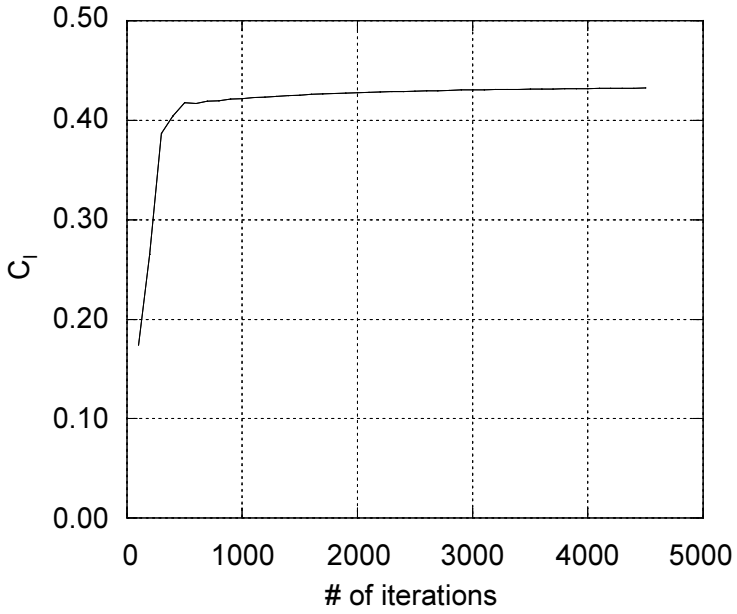


Figure 56: Convergence history of C_l with the high-fidelity model (RAE 2822, M=0.8, AoA=0.0 [deg]).

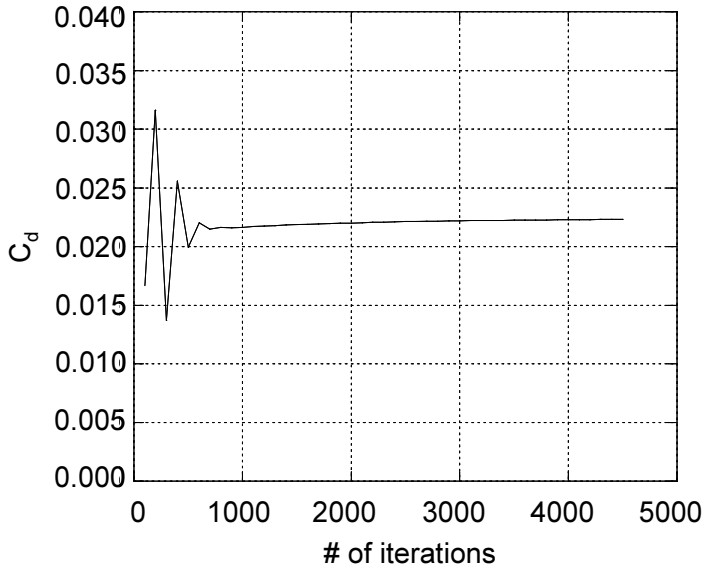


Figure 57: Convergence history of C_d with the high-fidelity model (RAE 2822, M=0.8, AoA=0.0 [deg]).

5.1.1.2 The derivative solver

In derivative-based optimizations, obtaining accurate derivatives in a shorter time is very important. In order to compute derivatives, the method proposed in Section 4.2 is implemented. For computing derivatives of nodes with respect to shape parameters, TAPENADE is applied to a code using the spring-analogy method because unstructured mesh is used in this case. For computing the derivatives of aerodynamics with respect to nodes, TAPENADE is applied to NSC2KE, and the generated derivative code is modified so that it does not require huge memory size.

Table 11 presents the comparisons of the derivatives of the AD and those of the FD with the low-fidelity model. In the FD method, step sizes $d\beta_i (i = 1, ..., 21)$ are set to 0.01, 0.001, and 0.0001. In Table 11, one can see some numerical discrepancies between those computed by the AD and those computed by the FD. However, as points $(\frac{\partial C_l}{\partial \beta_i}, \frac{\partial C_d}{\partial \beta_i})(i = 1, ..., 21)$, computed with the AD and the FD, are superimposed in a same graph shown in Figure 58 in order to check the matching between the AD and FD, one can confirm that they compute similar derivatives, especially when the step size is either 0.01 or 0.001. Here, the AD takes 150 seconds to compute all the derivatives while the FD takes 436 seconds for each step size, indicating that the derivative code works accurately and that the AD computes derivatives more accurately and more rapidly than the FD.

Table 11: Derivatives computed with the AD and the FD with the low-fidelity model (RAE 2822, M=0.8, α=0.0 [deg]).

	$\frac{\partial C_l}{\partial \beta_i}$				$\frac{\partial C_d}{\partial \beta_i}$			
i	AD	FD(dβ_i=0.1)	FD(dβ_i=0.01)	FD(dβ_i=0.001)	AD	FD(dβ_i=0.1)	FD(dβ_i=0.01)	FD(dβ_i=0.001)
1	-0.1068	-0.1145	-0.1157	-0.1157	-0.0061	-0.0090	-0.0090	-0.0090
2	-0.0128	-0.0126	-0.0131	-0.0131	0.0055	0.0049	0.0049	0.0049
3	-0.8868	-0.9141	-0.9136	-0.9136	-0.0007	0.0024	0.0025	0.0025
4	-3.5528	-3.4588	-3.5167	-3.5167	0.3504	0.4420	0.3986	0.3986
5	-3.8593	-3.4402	-3.6099	-3.6099	0.4991	0.5770	0.5344	0.5344
6	-4.2287	-3.9376	-4.0424	-4.0424	0.1274	0.1569	0.1398	0.1398
7	-4.3021	-4.3737	-4.3718	-4.3718	-0.3167	-0.3139	-0.3185	-0.3185
8	-3.9076	-4.0617	-4.0518	-4.0518	-0.3179	-0.3209	-0.3223	-0.3223
9	-4.1991	-4.3581	-4.3242	-4.3242	-0.2410	-0.2409	-0.2407	-0.2407
10	-5.7856	-6.0622	-5.9372	-5.9372	-0.2967	-0.2969	-0.2968	-0.2968
11	0.0873	0.0964	0.0993	0.0993	-0.0270	-0.0276	-0.0278	-0.0278
12	-0.0256	-0.0236	-0.0236	-0.0236	0.0008	-0.0001	-0.0001	-0.0001
13	0.9536	0.9821	0.9807	0.9807	0.0474	0.0540	0.0535	0.0535
14	4.6441	4.6770	4.6710	4.6710	0.9167	1.0348	0.9505	0.9505
15	2.9367	2.7055	2.7491	2.7491	1.7984	1.8131	1.7927	1.7927
16	0.4708	-0.2737	0.1493	0.1493	1.7338	1.7635	1.7183	1.7183
17	0.0842	-0.7406	-0.2235	-0.2235	0.9245	0.9620	0.9125	0.9125
18	4.8012	3.3613	4.5488	4.5488	0.1868	0.1830	0.1752	0.1752
19	13.4143	11.1902	13.3411	13.3411	-0.0842	-0.1507	-0.0936	-0.0936
20	6.6518	6.8857	6.8714	6.8714	0.2127	0.2239	0.2171	0.2171
21	-42.3452	-37.0562	-42.9072	-42.9072	-2.2701	-1.8723	-2.2451	-2.2451

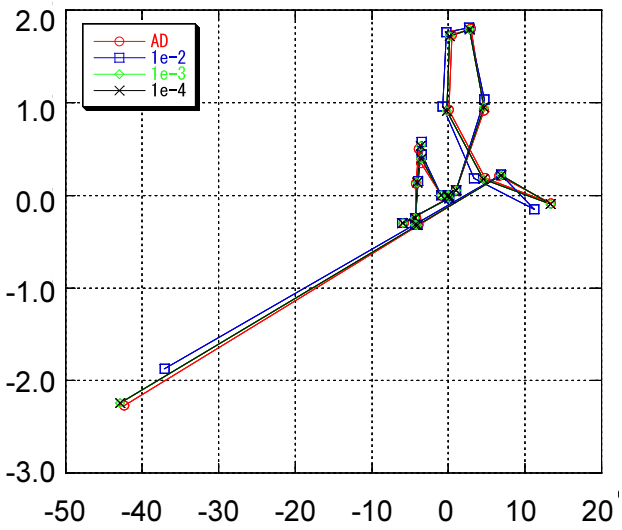


Figure 58: Comparison of the derivatives of the AD with those of the FD with the low-fidelity model (RAE 2822, M=0.8, α=0.0 [deg]).

The derivatives of the high-fidelity model are also computed. The step sizes used in the FD are 0.01, 0.001, and 0.0001. Table 12 and Figure 59 summarize the results of the comparison. Figure 59 indicates that the FD method with the step size of 0.01 is not suitable in this case. Here, the AD takes 1,114 seconds to compute all the derivatives while the FD takes 3,186 seconds for each step size. Again, one can conclude that the AD works well and that it computes derivatives more accurately and more rapidly than the FD.

Table 12: Derivatives computed with the AD and the FD with the high-fidelity model (RAE 2822, M=0.8, α=0.0 [deg]).

	$\frac{\partial C_l}{\partial \beta_i}$				$\frac{\partial C_d}{\partial \beta_i}$			
i	AD	FD(dβ_i=0.1)	FD(dβ_i=0.01)	FD(dβ_i=0.001)	AD	FD(dβ_i=0.1)	FD(dβ_i=0.01)	FD(dβ_i=0.001)
1	-0.1276	-0.1287	-0.1292	-0.1299	-0.0129	-0.0124	-0.0131	-0.0132
2	-0.0281	-0.0296	-0.0293	-0.0308	-0.0008	-0.0008	-0.0009	-0.0007
3	-0.7707	-0.8022	-0.7987	-0.8092	0.0067	0.0057	0.0047	0.0024
4	-2.4245	-1.7171	-2.5625	-2.6796	0.4878	0.8880	0.5233	0.4843
5	-1.8285	-1.3276	-2.1926	-2.4284	0.6497	1.0320	0.6786	0.6333
6	-2.3284	-2.3672	-2.8524	-3.0141	0.2121	0.5566	0.2118	0.1818
7	-3.3664	-3.9475	-3.7497	-3.8176	-0.2809	-0.2519	-0.3005	-0.2992
8	-3.3995	-3.6545	-3.5653	-3.5642	-0.2629	-0.2527	-0.2753	-0.2741
9	-3.4740	-3.9215	-3.6966	-3.6945	-0.1816	-0.1795	-0.1946	-0.2018
10	-4.4719	-5.0746	-4.9204	-4.9078	-0.2122	-0.1719	-0.2346	-0.2498
11	0.0901	0.0926	0.0942	0.0891	-0.0314	-0.0302	-0.0314	-0.0326
12	0.0113	0.0156	0.0151	0.0165	-0.0018	-0.0017	-0.0017	-0.0019
13	1.1320	1.0869	1.0974	1.0850	0.0611	0.0609	0.0588	0.0552
14	5.4057	3.6217	4.8912	4.9652	1.0334	1.4542	1.0887	1.0089
15	1.1942	1.4207	1.1385	1.5314	1.6957	1.9148	1.7295	1.7326
16	-2.7513	-1.0254	-2.1682	-1.8869	1.5501	2.1216	1.6688	1.6289
17	-3.7326	-3.6916	-3.3041	-2.9203	0.7711	1.6812	0.8900	0.8425
18	-0.8892	-2.2966	-0.3811	-0.0419	-0.0611	0.3953	0.0366	0.0074
19	7.7479	2.9474	9.3282	8.9264	-0.4488	-0.1013	-0.2680	-0.3373
20	6.1321	4.9238	6.5594	6.5713	0.1463	0.2003	0.1799	0.1866
21	-31.1029	-38.7840	-34.2872	-33.7543	-1.5753	-0.7252	-1.7125	-1.7578

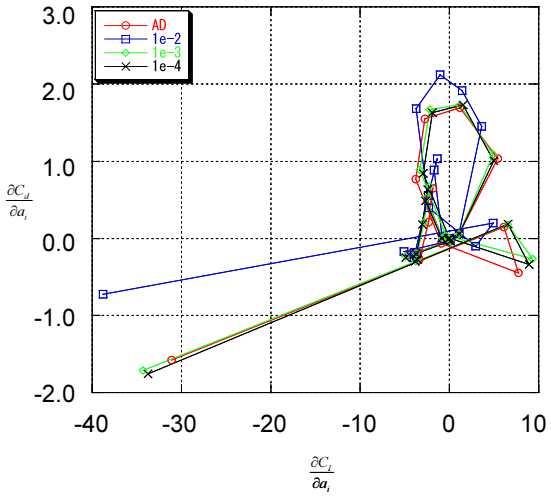


Figure 59: Comparison of the derivatives of the AD and those of the FD with the high-fidelity model (RAE 2822, M=0.8, α=0.0 [deg]).

To summarize the validations of the software, Table 13 presents the characteristics of the low- and high-fidelity models in the airfoil-design problem. The values in parentheses indicate the number of iterations required. In addition, in the high-fidelity model, the table shows that 176.22 [sec] is required for mesh adaptations and 122.20 [sec] for a flow calculation, each of which requires 1,000 and 2,500 iterations, respectively.

Table 13: Characteristics of the low- and high-fidelity models (airfoil).

		Low fidelity	High fidelity
		1,000-2,000 nodes	2,500-3,500 nodes, solution adapted
Flow solver	CPU time [sec]	16.32 (1,000)	176.22+122.20 (1000, 2500)
	Memory [MB]	5.4	6.6
Derivative code	CPU time [sec]	150.0 (500)	1114.0 (1,500)
	Memory [MB]	33.4	38.1

5.1.2 Optimization with the Low-Fidelity Model (Initial Airfoil: RAE 2822)

The airfoil shape is optimized with the low-fidelity model by means of the SQP in MATLAB. Here, the termination tolerance of the function value is set to 10^{-4}, that of the design variables 10^{-4}, and that of the constraint violation 10^{-2}. Default values are used for the remaining optimization parameters [2].

Figures 61, 62, 63, and 64 show the histories of drag coefficients, lift coefficients, and enclosed area, respectively. In the SQP, the quadratic model, which is expressed in Equation 64, is updated through the BFGS [18], [31], [38], [75], as shown in Figure 60. Therefore, while the quasi Hessian matrix is inaccurate due to the non linearity of the true model, intermediate quadratic models may not capture the behavior of the true model accurately, and this causes oscillation, as shown in Figures 61 and 62. Because of the inaccurate quadratic models, Figure 61 includes results that actually violate the constraints. Therefore, Figure 65, which shows the progression of the same optimization, is also provided. For the sake of clarity, Figure 65 only presents the objective function values that satisfy the given constraints. Viewed from left to right, each point represents the lowest C_d value up until the corresponding function call, which yields an overall trend similar to a step function. Initial and optimized aerodynamic coefficients, enclosed area, and CPU time, and the number of function calls required to obtain an optimized airfoil shape are summarized in Table 14.

$$\begin{aligned} f_{surrogate}\left(\mathbf{x}_n\right) &= f_{high}\left(\mathbf{x}_n\right) + \nabla f_{high}\left(\mathbf{x}_n\right)\left(\mathbf{x}-\mathbf{x}_n\right) \\ &+ \frac{1}{2}\left(\mathbf{x}-\mathbf{x}_n\right)^T \mathbf{H}_n\left(\mathbf{x}_n\right)\left(\mathbf{x}-\mathbf{x}_n\right) \end{aligned} \tag{64}$$

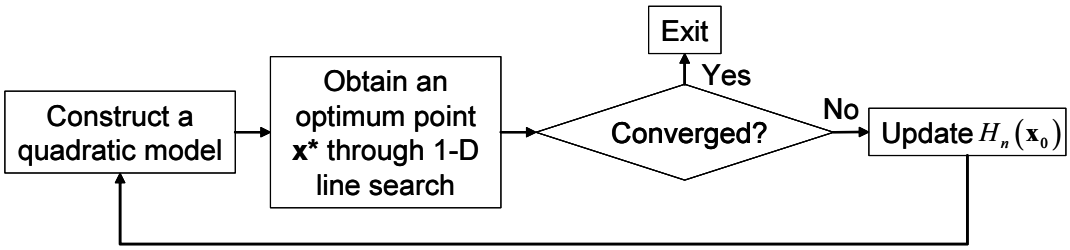


Figure 60: Flow chart of the SQP.

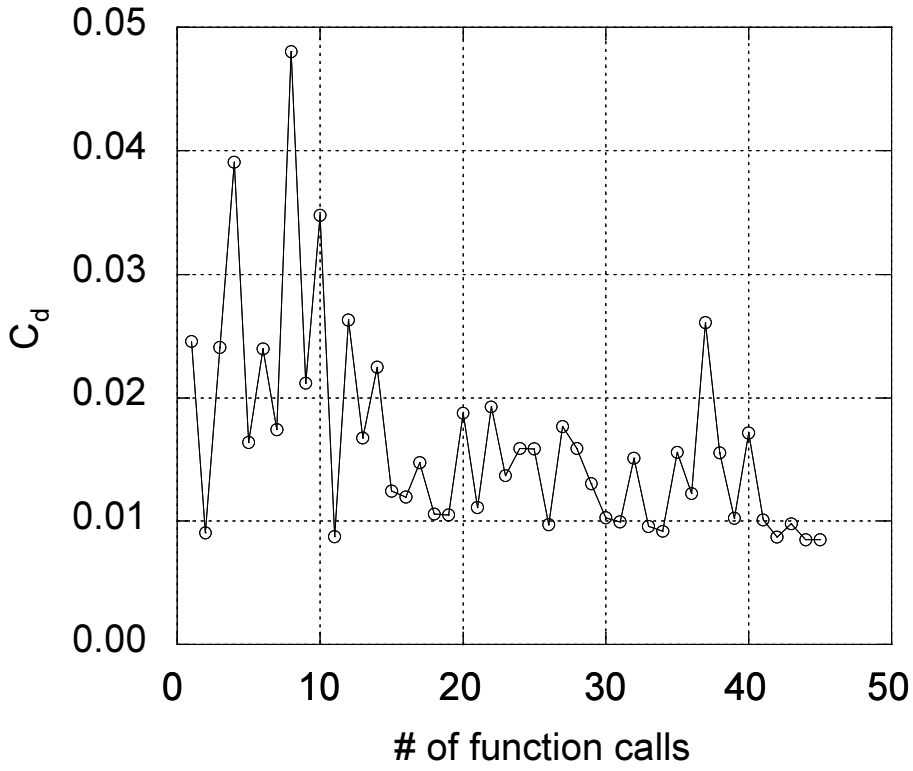


Figure 61: History of drag coefficients with the low-fidelity model.

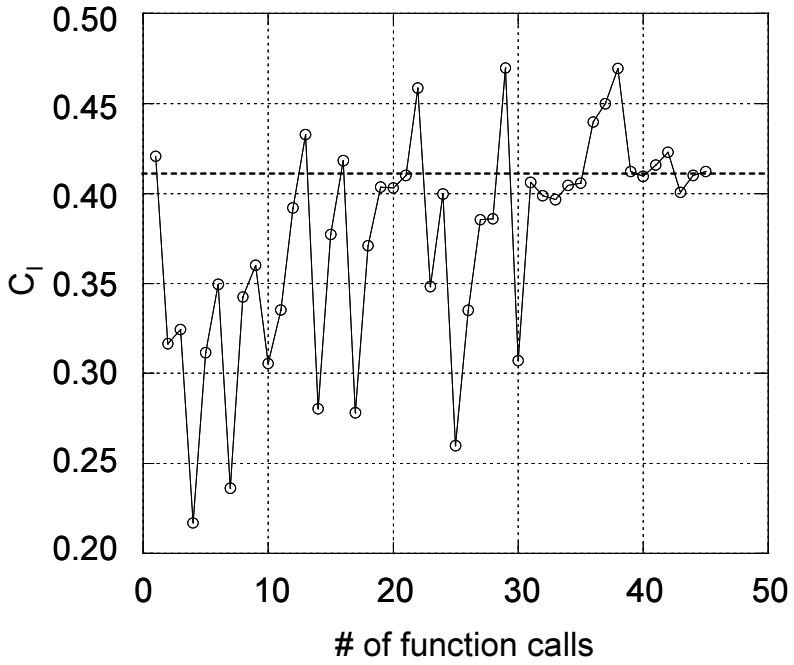


Figure 62: History of lift coefficients with the low-fidelity model.

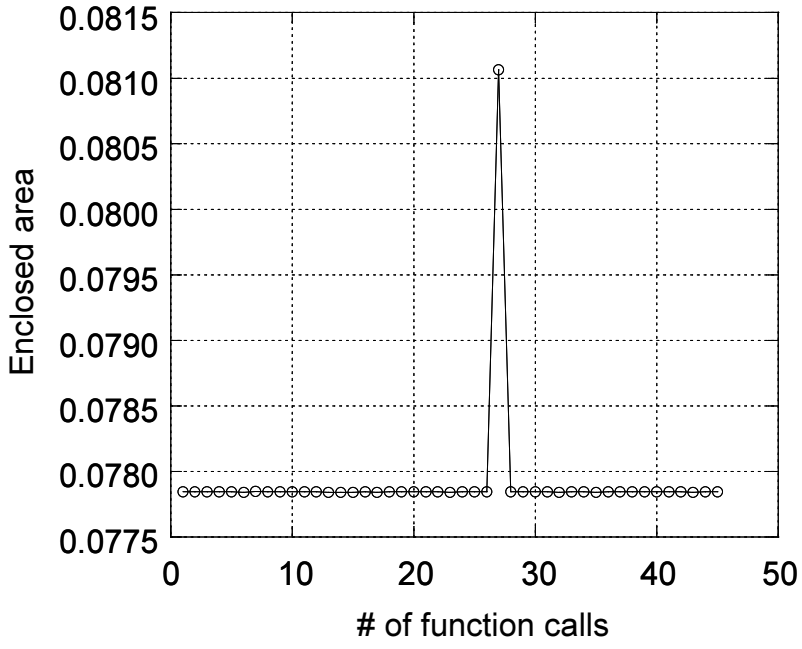


Figure 63: History of enclosed area with the low-fidelity model (1).

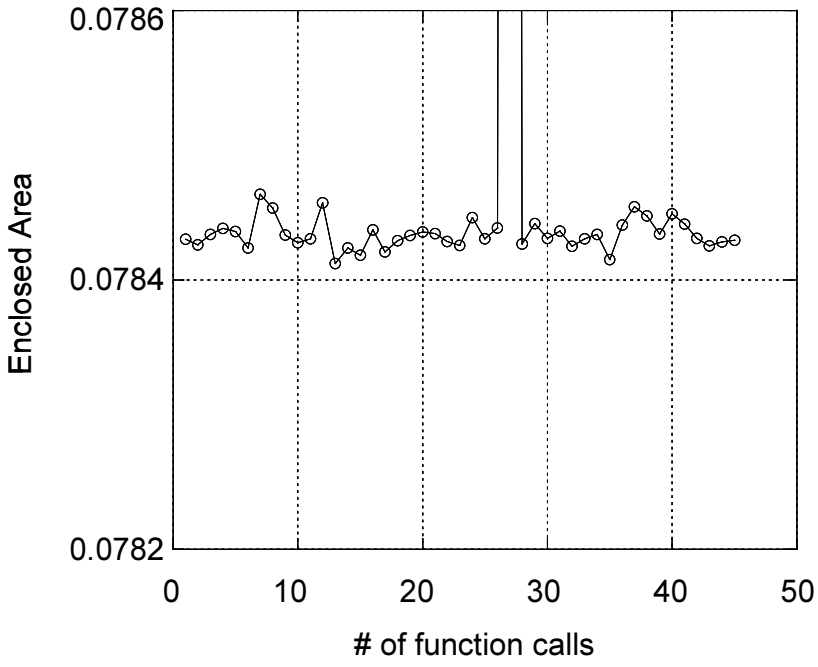


Figure 64: History of enclosed area with the low-fidelity model (2).

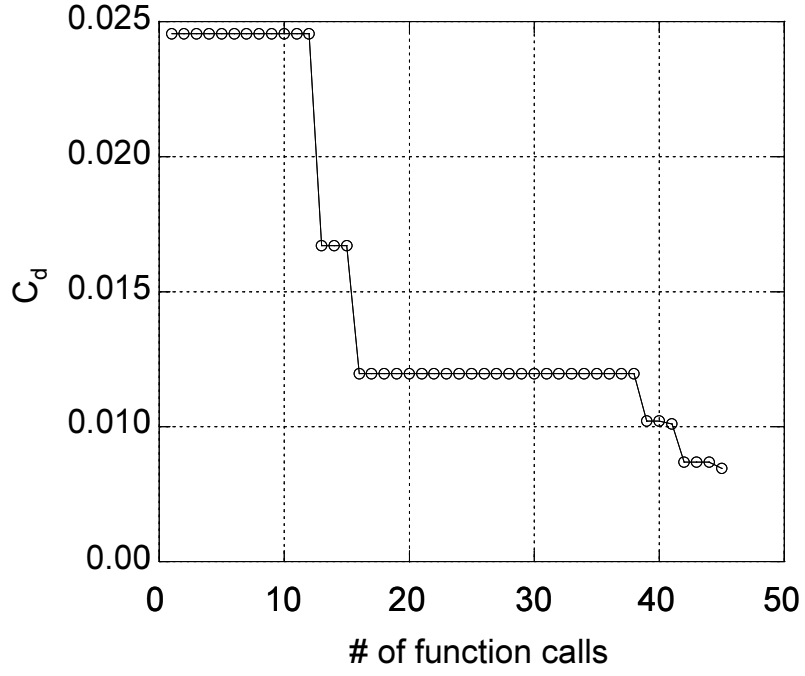


Figure 65: History of drag coefficients with the low-fidelity model (modified).

Table 14: Results with the low-fidelity model.

	C_d	C_l	Enclosed area	No. of f_{low}, ∇f_{low} calls	CPU time [sec]
Initial	0.0246	0.4210	0.0778	1, 0	16
Optimized	0.0085	0.4123	0.0778	45, 31	6,283

Figure 66 shows the initial and optimized airfoils, and Figure 67 the pressure coefficients around these airfoils. From Figures 66 and 67, one can conclude that flattening the upper surface of the airfoil mitigates the magnitude of the shock wave, facilitating drag reduction.

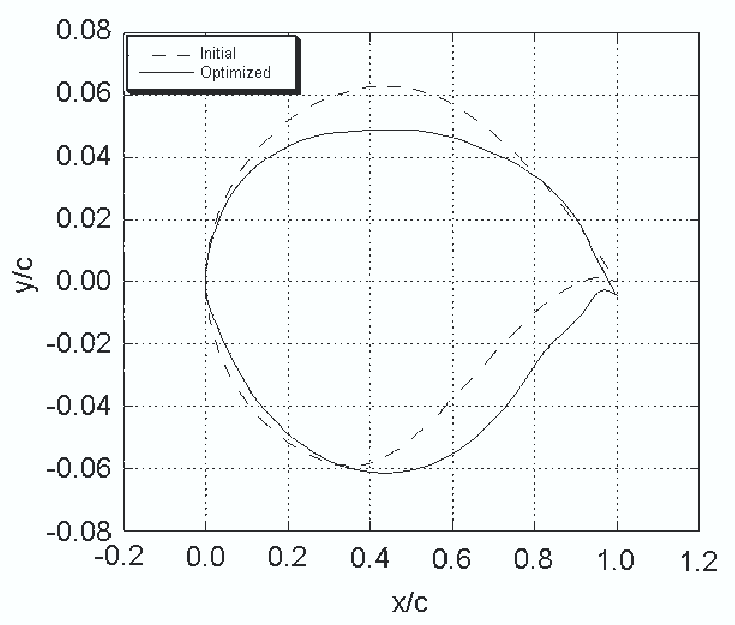


Figure 66: Initial and optimized airfoil shapes with the low-fidelity model.

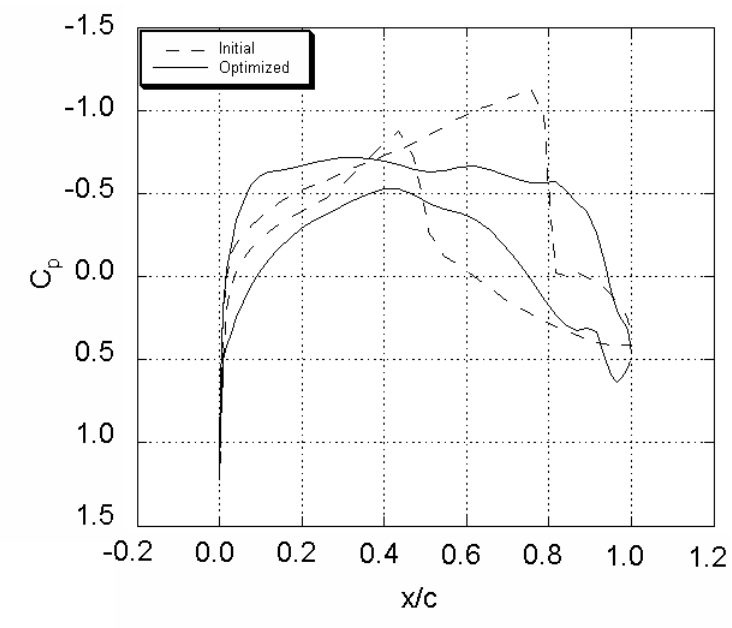


Figure 67: Distributions of pressure coefficients around the initial and optimized airfoils with the low-fidelity model.

The airfoil optimized with the low-fidelity model is analyzed with the high-fidelity model. In this case, C_d is 0.0057, and C_l is 0.4566, and an additional 359 seconds are required. Although the constraint of the lift coefficient set for the high-fidelity model ($C_l > C_{l,initial} = 0.4293$) is satisfied, this result is coincidental because the high-fidelity model computes a different C_l value than the low-fidelity model, and it may violate the constraint for C_l. In Figure 68, the distributions of pressure coefficients around the optimized airfoil analyzed with the low-fidelity model only (Low-Low) and those analyzed with the high-fidelity model at the very end (Low-High) are shown. The Low-High distribution curve show that shock waves occur on the upper and lower surface of the airfoil, which is not captured with the low-fidelity model only.

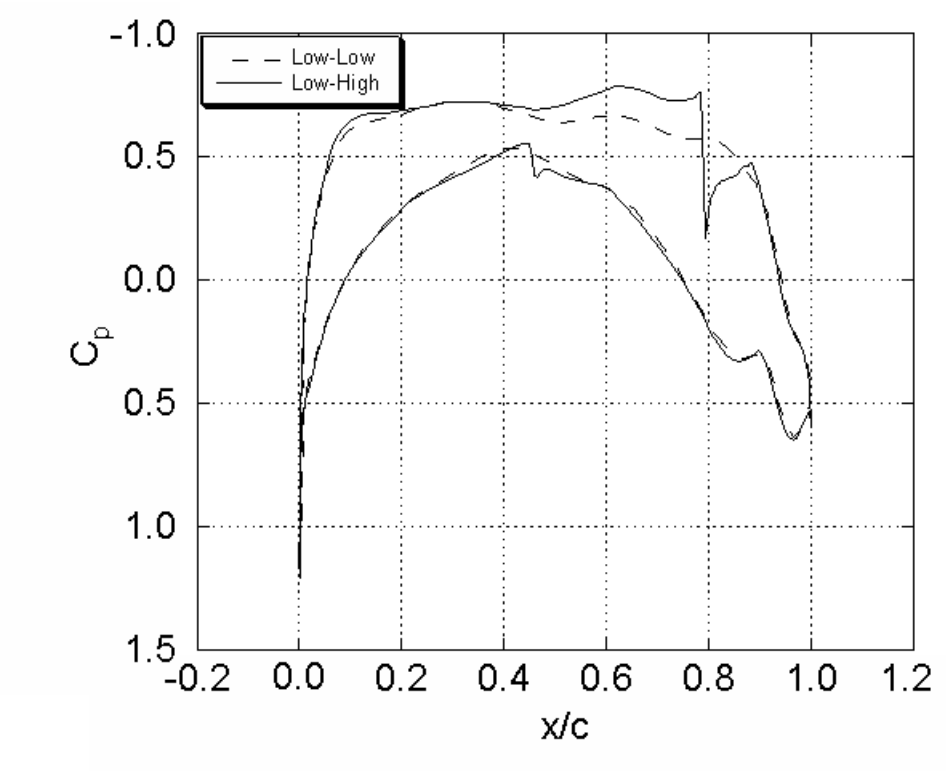


Figure 68: Distributions of pressure coefficients calculated by the low- and high-fidelity models around the airfoils optimized with the low-fidelity model.

5.1.3 Optimization with the High-Fidelity Model (Initial Airfoil: RAE 2822)

The airfoil shape is optimized with the high-fidelity model by means of the SQP in MATLAB. Figures 69, 70, and 71 show the histories of drag coefficients, lift coefficients, and enclosed area, respectively. Since Figure 69 includes results that actually violate the constraints, Figure 72, which shows the progression of the same optimization, is also provided. For the sake of clarity, Figure 72 only presents the objective function values that satisfy the given constraints. Viewed from left to right, each point represents the lowest C_d value up until the corresponding function call, which yields an overall trend similar to a step function. Initial and optimized aerodynamic coefficients, enclosed area, and CPU time, and the number of function calls required

to obtain an optimized airfoil shape are summarized in Table 15.

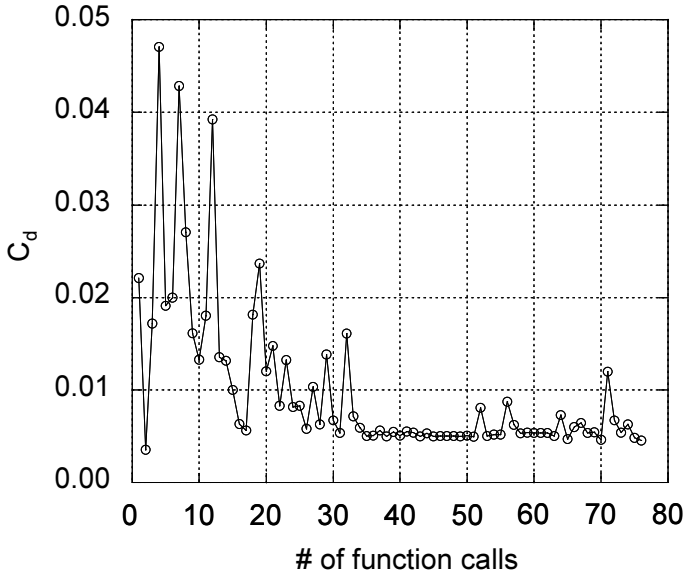


Figure 69: History of drag coefficients with the high-fidelity model.

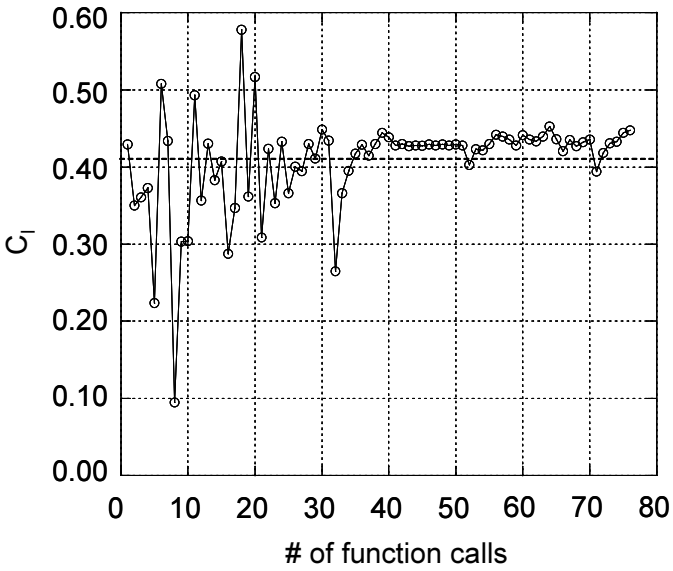


Figure 70: History of lift coefficients with the high-fidelity model.

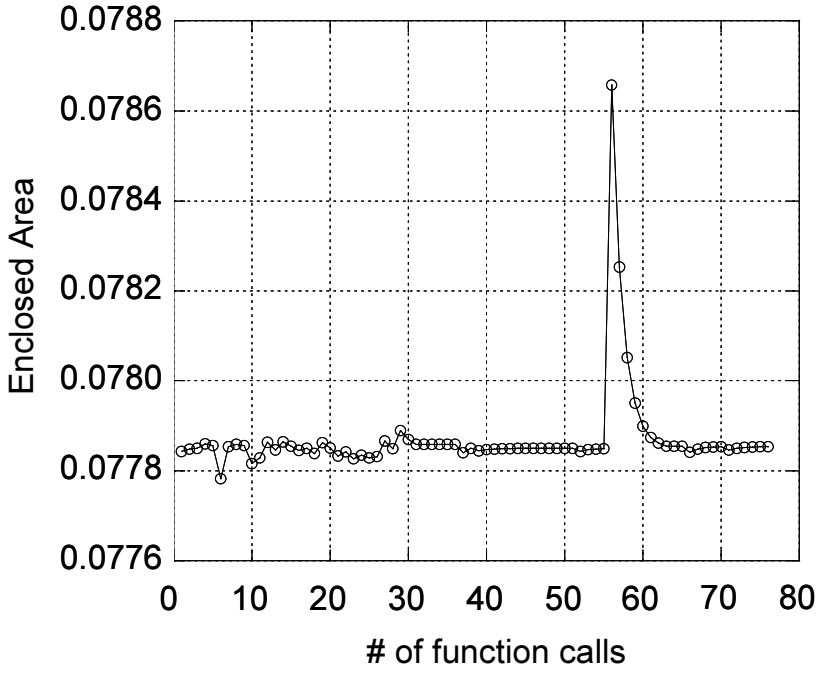


Figure 71: History of enclosed area with the high-fidelity model.

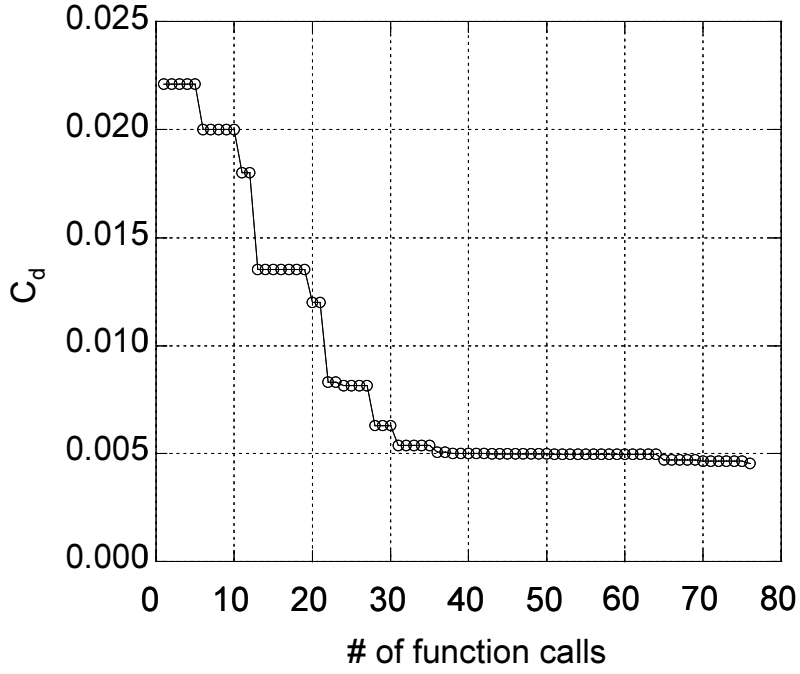


Figure 72: History of drag coefficients with the high-fidelity model (modified).

Table 15: Results with the high-fidelity model.

	C_d	C_l	Enclosed area	No. of f_{low}, ∇f_{low} calls	CPU time [sec]
Initial	0.0221	0.4293	0.0778	1, 0	359
Optimized	0.0045	0.4478	0.0779	76, 28	58,334

Figure 73 shows the initial and optimized airfoils, and Figure 74 shows the pressure coefficient around these airfoils. From Figures 73 and 74, one can conclude that flattening the upper surface of the airfoil and tailoring a square leading edge on the upper surface mitigate the magnitude of the shock wave, facilitating drag reduction. These characteristics, which are exactly the same as those of supercritical airfoils, developed by Whitcomb in the 1960s [85], have been employed on virtually all modern transonic airplanes. The square leading edge on the upper surface could not be created through optimization with the low-fidelity model because it did not have enough mesh resolution around the leading edge.

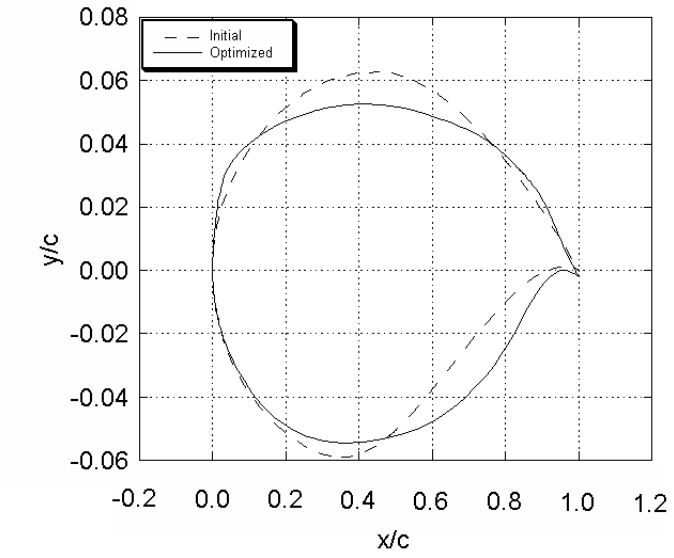


Figure 73: Initial and optimized airfoil shapes with the high-fidelity model.

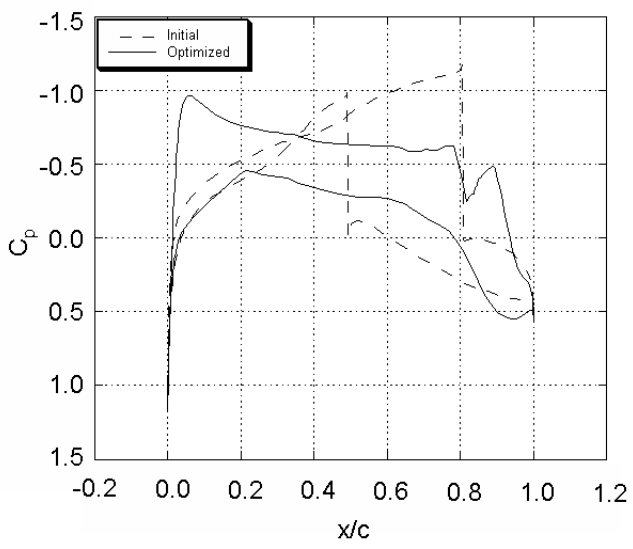


Figure 74: Distributions of the pressure coefficients around the initial and optimized airfoils with the high-fidelity model.

5.1.4 Optimization with the Robust AMF (Initial Airfoil: RAE 2822)

The airfoil shape is optimized with the Robust AMF. Here, the termination tolerance of the function value is set to 10^{-4} (Equation 29), that of the design variables 10^{-4} (Equation 30), and that of the constraint violation in the governing equation of the trust region ratio 10^{-2}. In addition, if a candidate design point $\mathbf{x}_c$ is rejected three consecutive times, the best design point that the Robust AMF has found is treated as the optimized solution, and the Robust AMF is terminated in this study. The termination is justified because the size of the trust region is only 1.6% (0.25^3) of the size of the entire domain after three rejections of $\mathbf{x}_c$, so probably one can not expect any better solution with the Robust AMF.

Table 16 shows a convergence history with the Robust AMF. In the second iteration, the Robust AMF finds a design point that actually violates constraints by 0.0082, but this is still within the tolerance of 10^{-2}. Therefore, as seen in the analytical constraint problem in Section 4.3, one can expect that the Robust AMF may

find a better answer after the third iteration because new surrogate functions may be created near the true optimum design point. However, since the degree of violation is over the tolerance of 10^{-2} after the third iteration, other design points are rejected, and the Robust AMF is terminated due to the three consecutive rejections of the candidate design point in this problem. Thus, the design point, which is found in the second iteration, is treated as the optimized design point. Termination of the optimization takes 34,948 seconds.

Table 16: Convergence history with the Robust AMF.

Iter	$f_{high}(\mathbf{x}_n)$	$g_{high}(\mathbf{x}_n)$		$f_{high}(\mathbf{x}_c)$	$g_{high}(\mathbf{x}_c)$		No. of function calls			
	C_d	C_l	Enclosed area	C_d	C_l	Enclosed area	f_{high},	f_{low},	∇f_{low},	∇f_{low}
0	0.0221	0.0000	0.0000				1,	1,	0,	0
1	0.0073	-0.0646	0.0000	0.0073	-0.0646	0.0000	2,	127,	1,	25
2	0.0049	0.0082	0.0000	0.0049	0.0082	0.0000	3,	173,	2,	34
3	0.0049	0.0082	0.0000	0.0066	0.2538	0.0000	4,	190,	3,	49
4	0.0049	0.0082	0.0000	0.0048	0.0951	0.0000	5,	210,	3,	63
5	0.0049	0.0082	0.0000	0.0052	0.0136	0.0000	6,	228,	3,	76

Figure 75 depicts the initial and optimized airfoils, and Figure 76 shows the distributions of the pressure coefficients around them. From Figures 75 and 76, one can conclude that the Robust AMF can indeed include the characteristics of the high-fidelity model because the optimized airfoil with the Robust AMF has not only a flat surface but also a square leading edge on the upper surface of the airfoil, which only the high-fidelity model can capture. However, the airfoil is not optimized in the Robust AMF as it is in the SQP with the high-fidelity model, in which the airfoil has the sharp suction peak at the leading edge, as seen in Figure 74. Thus, C_d of the optimized airfoil with the Robust AMF is slightly higher than that of the optimized

airfoil with only the high-fidelity model.

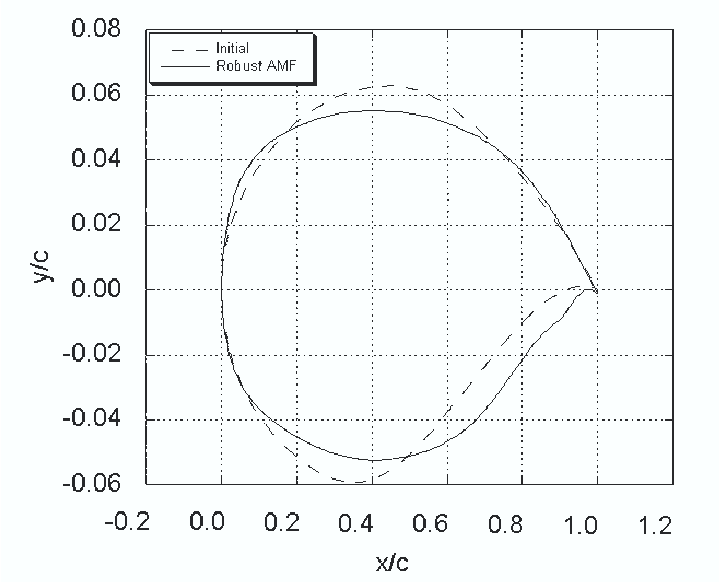


Figure 75: Initial and optimized airfoil shapes with the Robust AMF.

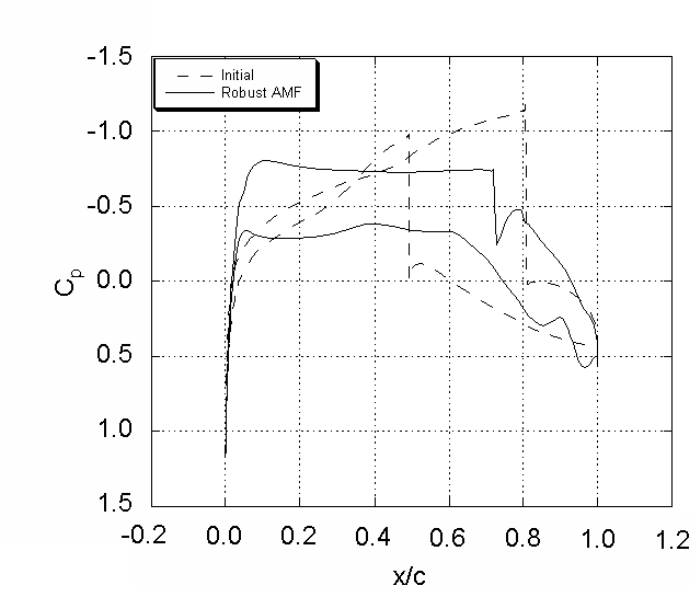


Figure 76: Distributions of the pressure coefficients around the initial and optimized airfoils with the Robust AMF.

5.1.5 Optimization with the Original AMF (Initial Airfoil: RAE 2822)

The airfoil shape is optimized with the original AMF. Here, the termination tolerance of the function value is set to 10^{-4} (Equation 29), and that of the design variables is set to 10^{-4} (Equation 30). Table 17 shows a convergence history with the original AMF. Since the original AMF does not accept any violations by candidate design point $\mathbf{x}_c$, it is slow at finding the optimized point, which is also observed in the simple analytical problem in 3.2.2. Fortunately, the original AMF finds the optimized design point within the feasible domain in this study. However, the value of the objective function at the optimized design point is higher than that with the Robust AMF. The optimized answer is obtained in 50,584 seconds.

Table 17: Convergence history with the original AMF.

Iter	$f_{high}(\mathbf{x}_n)$	$g_{high}(\mathbf{x}_n)$		$f_{high}(\mathbf{x}_c)$	$g_{high}(\mathbf{x}_c)$		No. of function calls			
	C_d	C_l	Enclosed area	C_d	C_l	Enclosed area	f_{high},	f_{low},	∇f_{low},	∇f_{low}
0	0.0221	0.0000	0.0000				1,	1,	0,	0
1	0.0073	-0.0646	0.0000	0.0073	-0.0646	0.0000	2,	127,	1,	25
2	0.0073	-0.0646	0.0000	0.0049	0.0082	0.0000	3,	173,	2,	34
3	0.0073	-0.0646	0.0000	0.0060	0.0055	0.0000	4,	220,	2,	49
4	0.0073	-0.0646	0.0000	0.0049	0.0370	0.0000	5,	286,	2,	72
5	0.0073	-0.0646	0.0000	0.0062	0.0293	-0.0003	6,	291,	2,	75
6	0.0065	-0.0440	0.0000	0.0065	-0.0440	0.0000	7,	294,	2,	77
7	0.0065	-0.0440	0.0000	0.0070	0.0321	-0.0002	8,	297,	3,	81
8	0.0065	-0.0440	0.0000	0.0066	-0.0167	-0.0001	9,	299,	3,	83
9	0.0065	-0.0440	0.0000	0.0065	-0.0466	0.0000	10,	304,	3,	84
10	0.0065	-0.0440	0.0000	0.0065	-0.0466	0.0000	11,	307,	3,	85
11	0.0065	-0.0440	0.0000	0.0065	-0.0457	0.0000	12,	311,	3,	86
12	0.0065	-0.0440	0.0000	0.0065	-0.0457	0.0000	13,	313,	3,	87
13	0.0065	-0.0440	0.0000	0.0065	-0.0442	0.0000	14,	315,	3,	88
14	0.0064	-0.0418	0.0000	0.0064	-0.0418	0.0000	15,	318,	3,	89

Figure 77 shows the initial and optimized airfoils, and Figure 78 shows the distributions of the pressure coefficients around these airfoils. Even though the drag is higher than that obtained with the Robust AMF or the SQP with the high-fidelity model, because of the flat lower surface, lift is the highest among all the results obtained with the other optimization methods.

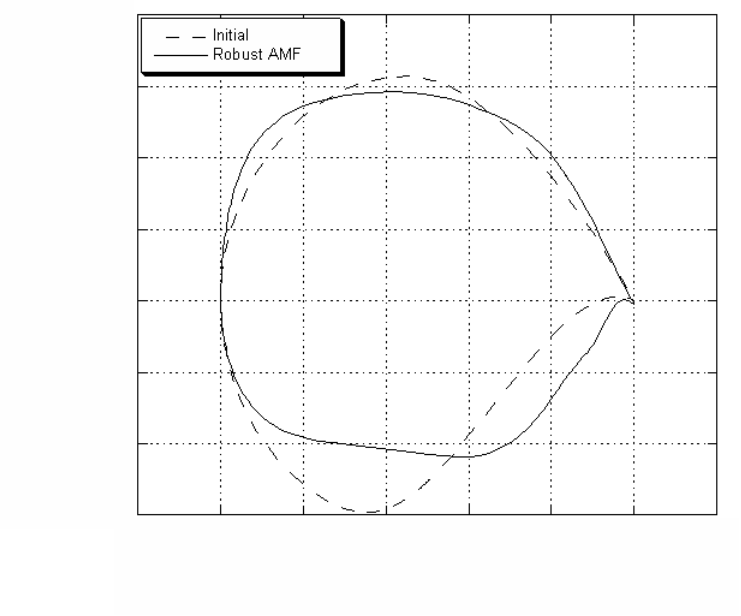


Figure 77: Initial and optimized airfoil shapes with the original AMF.

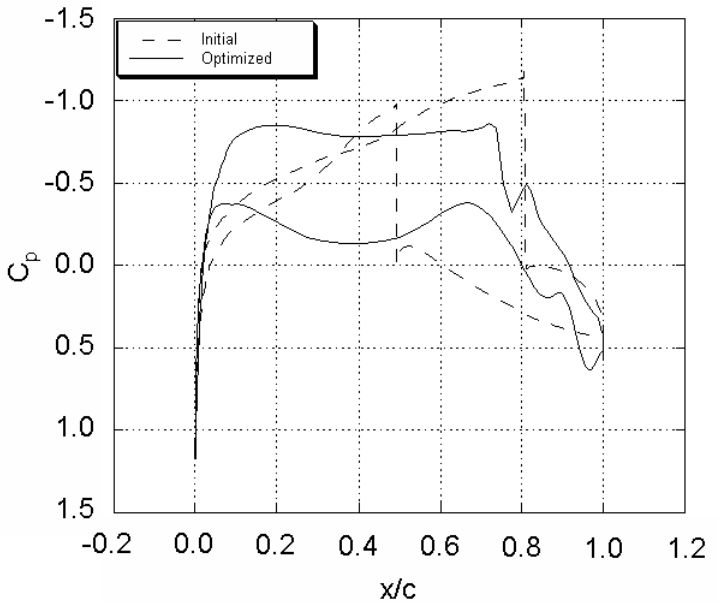


Figure 78: Distributions of the pressure coefficients around the initial and optimized airfoils with the original AMF.

5.1.6 Summary of the Design of an Airfoil in the Transonic Speed Regime (Initial Airfoil: RAE 2822)

All optimization results obtained from the different optimization methods for the design of an airfoil in the transonic speed regime are summarized in Table 18. In columns "C_l" and "Enclosed area," the values in parentheses represent the degree of violation of the constraints, and in column "CPU time," the values in parentheses represent non-dimensional CPU time normalized by the CPU time required in the SQP with the high-fidelity model.

From Table 18, the following remarks can be deduced:

- The SQP with the high-fidelity model works the best in terms of the optimized value, but it takes the longest in CPU time.
- The SQP with the low-fidelity model works well in this study. However, this finding is coincidental because the constraints may be violated in different optimization problems if the airfoil is optimized only with the low-fidelity model and analyzed with the high-fidelity model at the end.
- The Robust is competitive with the SQP with the high-fidelity model. In a comparison of the results by the SQP with the high-fidelity model, the required CPU time is 40% shorter although the optimized C_d value is 4 counts (4/10000) higher.
- If a slight violation of the constraint is allowed in its final design, the Robust AMF works better than the original AMF in terms of the final aerodynamics and the required CPU time because the Robust AMF allows the optimizer wider exploration.
- The original AMF used in this study is not significantly beneficial. In a comparison of the results by the SQP with the high-fidelity model, the CPU time required is 14% lower, and the C_d value obtained is 19 counts (19/10000) higher.

Table 18: Comparison of results with different optimization methods.

	C_d	C_l	Enclosed area	No. of f_{high}, f_{low}, ∇f_{high}, ∇f_{low} calls	CPU time [sec]
SQP with f_{high}	0.0045	0.4478	0.0779	76, 0, 28, 0	58,334
		(-0.0185)	(-0.0001)		(1.00)
SQP with f_{low}	0.0057	0.4566	0.0778	1, 45, 0, 31	6,642
		(-0.0273)	(0.0000)		(0.11)
Robust AMF	0.0049	0.4211	0.0778	6, 228, 3, 76	34,948
		(+0.0082)	(0.0000)		(0.60)
Original AMF	0.0064	0.4711	0.0779	15, 381, 3, 89	50,584
		(-0.0418)	(-0.0001)		(0.86)
Initial	0.0221	0.4293	0.0778	1, 0, 0, 0	359

Optimized airfoil shapes with different optimization methods and distributions of their pressure coefficients are compared in Figures 79 and 80, respectively. From Figures 79 and 80, the following remarks can be deduced:

- Airfoil shapes optimized with the SQP with the high-fidelity model and the Robust AMF are closest among all the other airfoils.
- Distributions of pressure coefficients are divergent. However, the distributions computed by the SQP with the high-fidelity model and the Robust AMF are closer to each other than they are to the others. If the robust AMF can capture the suction peak more accurately, the distribution computed with the Robust AMF may be closer to that computed by the SQP with the high-fidelity model.
- The original AMF apparently leads to a different optimized airfoil.

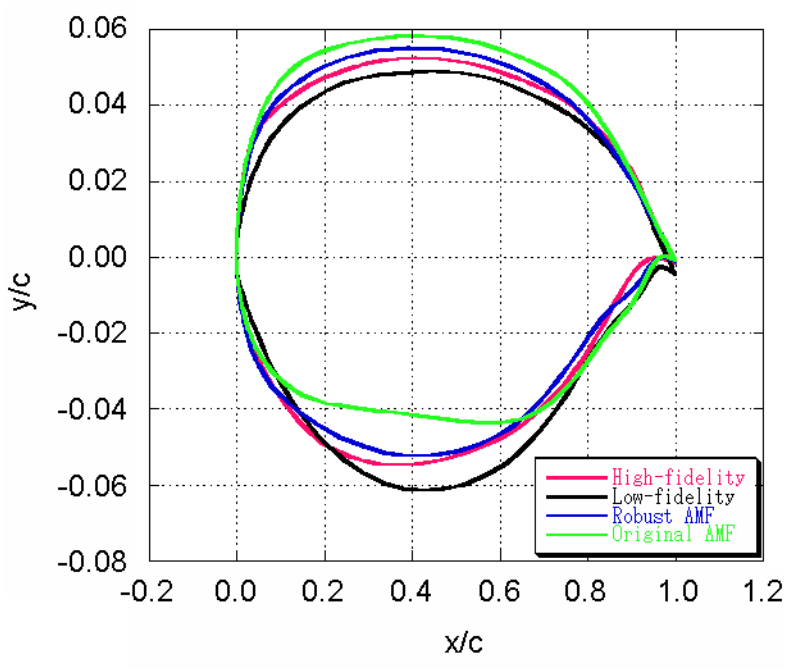


Figure 79: Optimized airfoil shapes with different optimization methods.

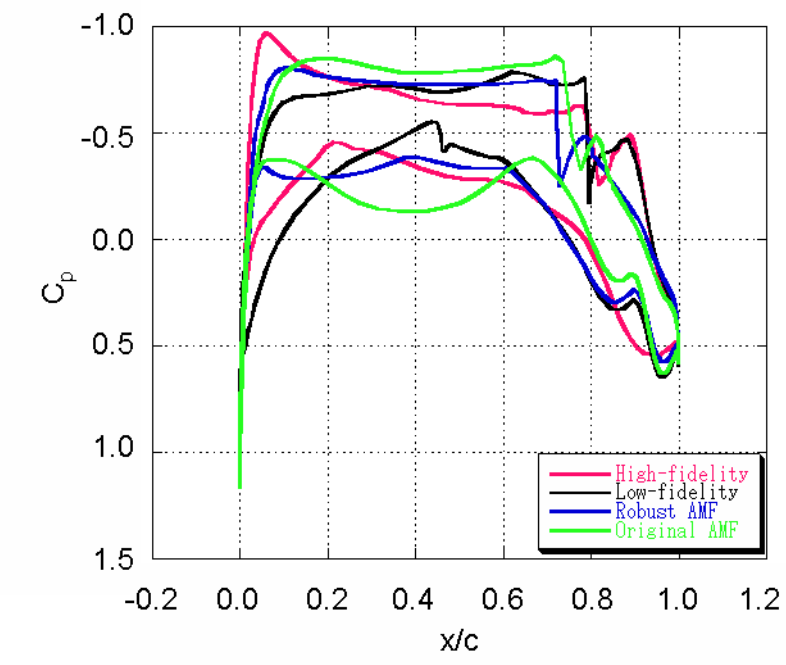


Figure 80: Distributions of the pressure coefficients around the optimized airfoils with different optimization methods.

5.1.7 Optimization (Initial Airfoil: RAE 5212)

This time, RAE 5212 is used as an initial airfoil. Everything in the problem setting, except the initial airfoil shape, is the same as it is in the RAE 2822 case.

Table 19 summarizes the results with different optimization methods. Figure 81 shows comparisons of the initial and optimized airfoil shapes, and Figure 82 shows their distributions of pressure coefficients.

Table 19: Comparison of results with different optimization methods.

	C_d	C_l	Enclosed area	No. of f_{high}, f_{low}, ∇f_{high}, ∇f_{low} calls	CPU time [sec]
SQP with f_{high}	0.0081	0.5723	0.0793	58, 0, 28, 0	47,544
		(-0.0029)	(0.0000)		(1.00)
Robust AMF	0.0095	0.5837	0.0793	5, 309, 2, 61	42,429
		(-0.0143)	(0.0000)		(0.89)
Original AMF	0.0095	0.5837	0.0793	5, 309, 2, 61	42,429
		(-0.0143)	(0.0000)		(0.89)
Initial	0.0329	0.5694	0.0793	1, 0, 0, 0	359

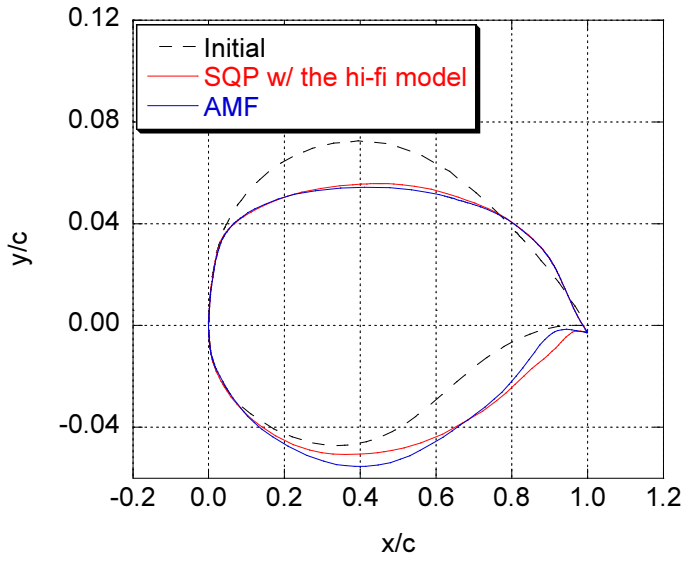


Figure 81: Initial and optimized airfoil shapes with different optimization methods.

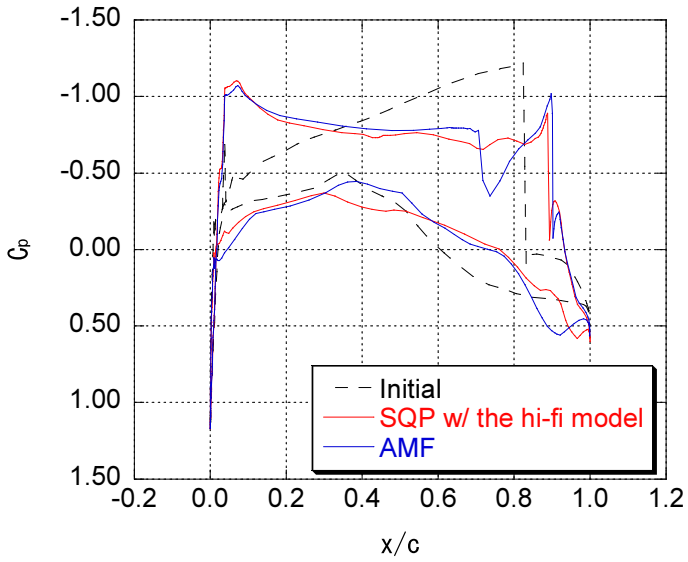


Figure 82: Distributions of the pressure coefficients around the initial and optimized airfoil with different optimization methods.

Table 19 shows all optimization methods succeeded in this optimization problem. Here, because constraints are never violated during the entire optimization process, the Robust AMF and the original AMF produce the same results. From Figures 81 and 82, the Robust AMF and the SQP with the high-fidelity models find similar airfoils. That is, the shock waves are mitigated as the upper surface of the airfoils are flattened.

Although the constraint are not violated, the Robust AMF with tightened constraint by the amount of $tolg$ is implemented because the Robust AMF can potentially violate the constraint at the end of the optimization. The result is summarized in Table 20, in which the result obtained under the original constraint is also presented.

Table 20: Comparison of results with different constraints in the Robust AMF.

	C_D	C_L	Volume	No. of f_{high},	f_{low},	∇f_{high},	∇f_{low}	calls	CPU time [sec]
$C_L > C_{L,initial}$	0.0095	0.5837	0.0793	5,	309,	2,	61		42,429
		(-0.0143)	(0.0000)						
$C_L > C_{L,initial} + tolg$	0.0101	0.6305	0.0793	5,	103,	2,	56		29,024
		(-0.0611)	(0.00000)						
Initial	0.0329	0.5694	0.0793	1,	0,	0,	0		226

As expected, the optimum design point moves more inside of the feasible region.

5.1.8 Optimization (Initial Airfoil: RAE 5214)

This time, RAE 5214 is used as an initial airfoil. Everything in the problem setting, except the initial airfoil shape, is the same as it is in the RAE 2822 case. Since RAE 5214 has much smaller drag but almost same lift as RAE 2822 at $M_\infty = 0.8$ with an angle of attack $\alpha = 0.0$ [deg] (i.e., RAE 5214 is more sophisticated airfoil than RAE 2822), optimizing RAE 5214 may be interesting.

Table 21 summarizes the results with different optimization methods. Figure 83 shows comparisons of the initial and optimized airfoil shapes, and Figure 84 shows their distributions of pressure coefficients.

Table 21: Comparison of results with different optimization methods.

	C_d	C_l	Enclosed area	No. of f_{high},	f_{low},	∇f_{high},	∇f_{low}	calls	CPU time [sec]
SQP with f_{high}	0.0050	0.4332	0.0633	44,	0,	23,	0		40,265
		(-0.0026)	(0.0000)						(1.00)
Robust AMF	0.0055	0.4354	0.0633	5,	92,	2,	64		33,331
		(-0.0048)	(0.0000)						(0.83)
Original AMF	0.0055	0.4354	0.0633	5,	92,	2,	64		33,331
		(-0.0048)	(0.0000)						(0.83)
Initial	0.0077	0.4306	0.0633	1,	0,	0,	0		359

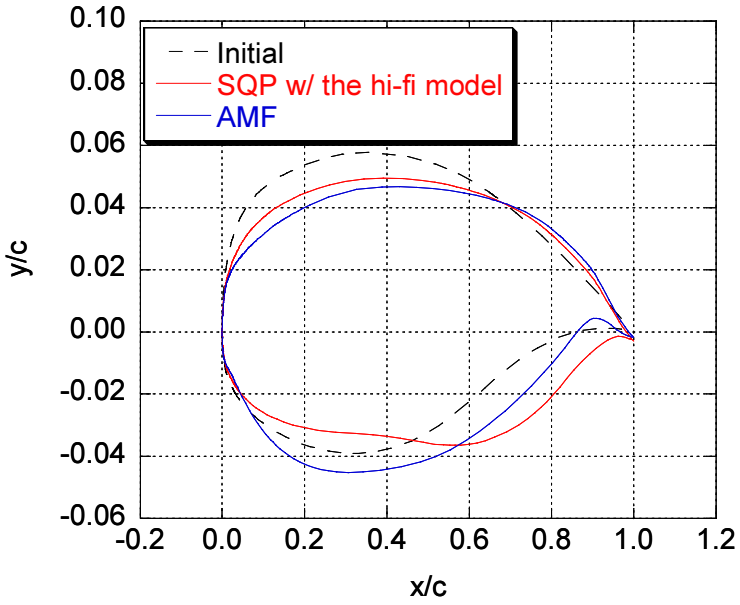


Figure 83: Initial and optimized airfoil shapes with different optimization methods.

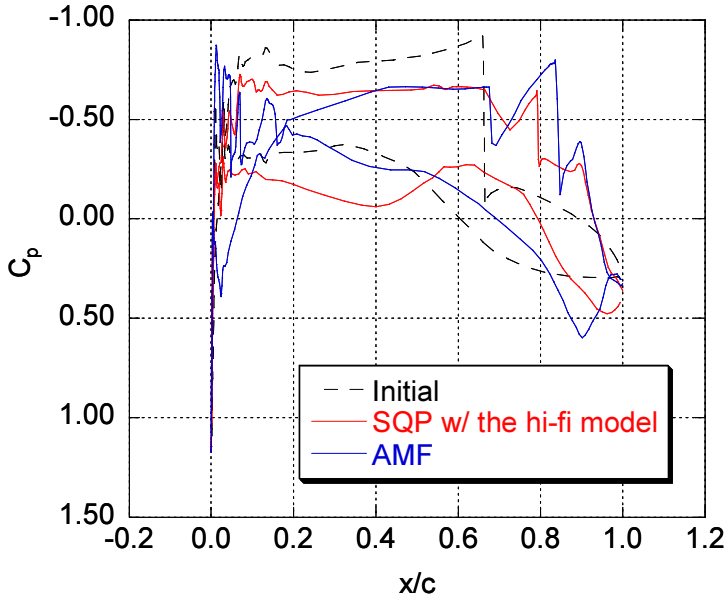


Figure 84: Distributions of the pressure coefficients around the initial and optimized airfoil with different optimization methods.

Figure 84 shows unrealistic distributions of the pressure coefficients at the upper

leading edge. In order to solve this problem, more grid adaptations were attempted to increase the grid resolution, but no significant improvements were observed. It was also confirmed that the oscillation observed remained present even after the first-order flux evaluation was performed as well as a reduction by 50% of the original CFL number. Furthermore, the distributions of pressure coefficients of the initial airfoil (RAE 5214) were plotted in Figure 85, and it was observed the oscillations are presented. Therefore, it was concluded that the root cause of these oscillations is inherent to the CFD solvers and is not attributable to the Robust AMF.

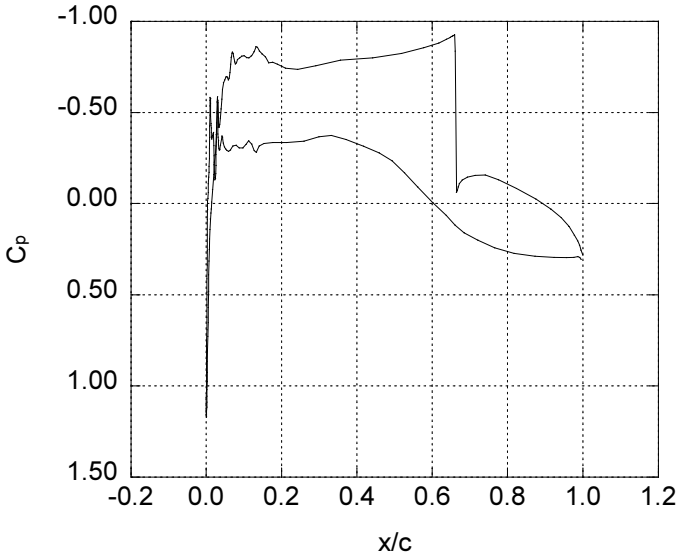


Figure 85: Distributions of the pressure coefficients around RAE 5214.

5.2 Supplemental Study in the Design of an Airfoil in the Transonic Speed Regime

In the previous section, the airfoil was optimized using several optimization methods. The study showed that the Robust AMF may finish the optimization in the infeasible region. Therefore, the author proposes following three methods to solve this problem:

- Use another optimization method after obtaining the optimum design point

with the Robust AMF.

- Change the angle of attack so that the airfoil optimized with the Robust AMF can satisfy the constraint.

- Tighten the constraint by the amount of $tolg$ (i.e., if the original constraint is $C_l > 0.1$ and $tolg = 0.01$, the new tightened constraint is $C_l > 0.1 + 0.01$).

The first method sequentially uses the Robust AMF and another optimization method that is stricter against the violation of constraints. Although it uses two different optimization methods, which may require more computational time, this method is effective for all kinds of violated constraints. The second method is effective when the violated constraint is a function of the angle of attack (i.e., C_l, C_d). The third method is effective for all kinds of violated constraints, but the optimization problem may be more difficult. In the following subsections, each method is implemented and discussed.

5.2.1 Use another optimization method after obtaining the optimum design point with the Robust AMF

The original AFM and SQP, which are stricter against the violation of constraints, are implemented after obtaining the optimum design point with the Robust AMF. Therefore, the initial design point for the second optimization method (i.e., the original AMF or SQP) is the optimized design point obtained by the Robust AMF. Table 22 summarizes the results. In Table 22, the values in parentheses represent the degree of violation of the constraint. Here, the same convergence criteria that are set in the optimization through the original AMF, the Robust AMF, and SQP in the previous section are used.

Table 22: Comparison of aerodynamics with different optimization methods.

	C_l	C_d	C_l/C_d	Additional CPU time
Initial	0.4293	0.0221	19.43	-
R-AMF	0.4211 (+0.0082)	0.0049	85.94	-
R-AMF→ Original AMF	0.4552 (-0.0259)	0.0057	79.86	$>$ 2 [day]
R-AMF→ SQP w/ the hi-fi model	0.4303 (-0.0010)	0.0050	86.06	61,011 [sec]

As expected, the original AMF and SQP find optimum design points in the feasible region. However, they consume significantly more additional CPU time (the optimization does not even converge in the R-AMF→ SQP w/ the hi-fi model case), which may be related to the high nonlinearity of the true model and the bad initial design point found by the Robust AMF. Thus, while this approach for addressing constraint violations does not appear promising in terms of CPU time, it yields a design point in the feasible region.

5.2.2 Change the angle of attack

Since the constraint C_l is a function of the angle of attack, one can manipulate C_l by changing the angle of attack.

The optimized airfoils are manually rotated so that they yield the nearly same C_l value (=0.4293) as the initial airfoil. Table 23 summarizes the results. In Table 23, the values in parentheses represent the degree of violation of the constraint.

Table 23: Comparison of aerodynamics with different optimization methods.

	C_l	C_d	C_l/C_d	$\alpha[deg]$
Initial	0.4293	0.0221	19.43	0.00
Robust AMF	0.4211 (+0.0082)	0.0049	85.94	0.00
	0.4294 (-0.0001)	0.0051	84.20	0.03
Original AMF	0.4711 (-0.0418)	0.0064	73.61	0.00
	0.4299 (-0.0006)	0.0058	74.12	-0.13
SQP w/ the hi-fi model	0.4478 (-0.0185)	0.0045	99.51	0.00
	0.4294 (-0.0001)	0.0043	99.86	-0.06

From the C_d and C_l/C_d values in Table 23, one can confirm that the airfoil optimized with the SQP with the high-fidelity model is the best in terms of aerodynamics. Furthermore, although the angle of attack is changed manually after the optimized airfoils are found this time, it is desirable to manipulate the angle of attack automatically at every iteration. Thus, this approach results in a feasible design point, so methods of automating the angle of attack should be explored in future work.

5.2.3 Tighten the constraint by the amount of *tolg*

In the Robust AMF, the constraint and $tolg$ were set to $C_l > C_{l,initial}$ and $tolg = 0.01$, and the Robust AMF is implemented under the new tightened constraint, $C_l >$

$C_{l,initial} + tolg = C_{l,initial} + 0.01$. The results are summarized in Table 24, in which the values in parentheses represent the degree of violation of the constraint. Here, the same convergence criteria that are set in the optimization through the Robust AMF in the previous section are used.

Table 24: Comparison of aerodynamics with different constraints in the Robust AMF.

	C_l	C_d	C_l/C_d	CPU time [sec]
Initial	0.4293	0.0221	19.43	359
$C_l > C_{l,intial}$	0.4211 (+0.0082)	0.0049	85.94	34,948
$C_l > C_{l,intial} + tolg$	0.4389 (-0.0096)	0.0052	84.40	21,921

The optimized airfoil by the Robust AMF with the tightened constraints has not only a smaller C_d value than the initial airfoil but also meets the constraint of guaranteed higher C_l value than the initial airfoil. The Robust AMF with tightened constraints does this in reasonable CPU time. Since this method is effective for all kinds of violated constraints, it could prove a promising method in any optimization method.

5.3 Design of a Wing in the Supersonic Speed Regime

The second optimization problem in aerospace engineering is the design of a wing in the supersonic speed regime. In supersonic aircraft, a double-delta wing whose center of lift moves from the back to the front as flight speed increases is desirable for the purpose of stability [66]. Therefore, the design of the double-delta wing is considered in this study.

5.3.1 Setting Up an Optimization Problem

To set up an optimization problem, let us first refer to a supersonic business jet under development. Figure 86 shows the Sukhoi-Gulfstream S-21, which has been under development since the 1990s, and Table 25 summarizes its technical specifications [58].

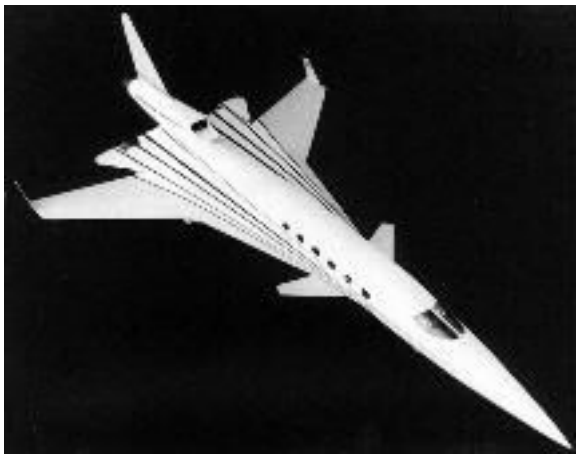


Figure 86: Sukhoi-Gulfstream S-21 [58].

Table 25: Technical specifications of the S-21 [58].

Wing span	65.34(19.92)	ft(m)
Length	132.84(40.5)	ft(m)
Wing LEX sweep	68	deg
Outer wing LE sweep	32	deg
Wing area	1,400(130)	$ft^2(m^2)$
Max. takeoff weight	106,920(48,500)	lb(kg)
Takeoff gross weight	105,820(48,000)	lb(kg)
Empty weight	49,737(22,560)	lb(kg)
Cruise speed, subsonic	631(1,015) M=0.95	mph(km/h)
Cruise speed, supersonic	1,320(2,125) M=2.0	mph(km/h)
Ceiling	60,696(185,00)	ft(m)
Range @ M=2.0	4,598(7,400)	mi(km)

Second, let us consider a situation in which an S-21 is flying horizontally at Mach 2.0 at 50,000 feet. In this case, the required lift coefficient based on the wing area is computed as follows:

$$\begin{aligned} C_L &= \frac{W}{\frac{1}{2}\rho V^2 S_{wing}} = \frac{W}{\frac{1}{2}\rho \left(M_\infty \sqrt{\gamma RT}\right)^2 S_{wing}} \\ &= \frac{106,920}{0.5 \times 3.6391 \times 10^{-4} \times \left(2.0 \times \sqrt{1.4 \times 1,716 \times 389.99}\right)^2 \times 1,400} \\ &= 0.1120. \end{aligned} \tag{65}$$

Third, let us create a baseline geometry. In reference [44], the aerodynamics of several cranked leading-edge wing-body configurations are experimentally analyzed and the results of wind tunnel experiments are available to the public. Therefore, one of the cranked leading-edge wing-body configurations in reference [44], which is geometrically most similar to that of the S-21, is selected as the planform of the baseline. However, due to limitations of the CFD analysis tool used in this study, only its wing part shown in Figure 87 is extracted from the original model and analyzed in computer simulations. In the baseline configuration, the twist angles at sections A, B, C, and D in Figure 87 are 6.0, 4.0, 3.0, and 0.0 [deg], respectively, so the initial baseline configuration has enough lift, and optimization can start from a feasible design point.

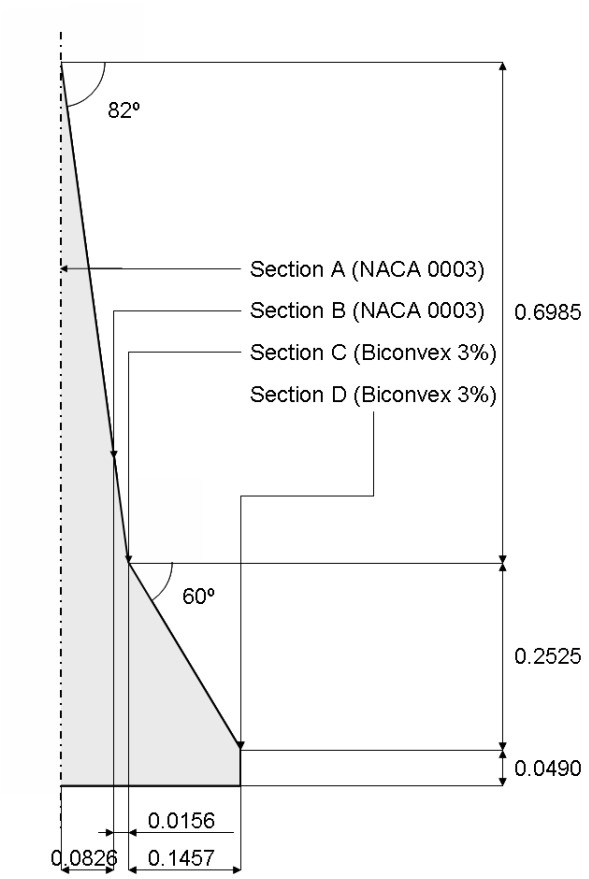


Figure 87: Baseline geometry.

In the optimization process, the thicknesses, the cambers, and the twist angles of sections A, B, C, and D change. The thickness and camber of each section are represented by Bezier curves [64], which are governed by seven variables in this study. In addition, in order to change the locations of the trailing edges, βx^{10} (β is a control variable, and x is non-dimensional chord length) is added to the Bezier curve for the camber in each section. Regarding the twist, each section is rotated around the trailing edge. Therefore, the total number of control variables is 64 (=4(7+7+1)+4). This setting is mathematically represented in Equation 66, whose supplemental information is summarized in Table 26. The method of generating an airfoil in each section is depicted in Figure 88.

$$\mathbf{p}(t) = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} (1-t)^{n-i} t^i \mathbf{B}_i, (0 < t < 1), n = 8. \tag{66}$$

Table 26: Locations of the control points of the Bezier curve for airfoil thickness and camber.

Thickness			Camber		
$\mathbf{B}_{thickness,i}$	x	y	$\mathbf{B}_{camber,i}$	x	y
0	0.000	0.000	0	0.000	0.000
1	0.001	$\beta_{1,16,31,46}$	1	0.125	$\beta_{8,23,38,53}$
2	0.050	$\beta_{2,17,32,47}$	2	0.250	$\beta_{9,24,39,54}$
3	0.200	$\beta_{3,18,33,48}$	3	0.375	$\beta_{10,25,40,55}$
4	0.400	$\beta_{4,19,34,49}$	4	0.500	$\beta_{11,26,41,56}$
5	0.500	$\beta_{5,20,35,50}$	5	0.625	$\beta_{12,27,42,57}$
6	0.700	$\beta_{6,21,36,51}$	6	0.750	$\beta_{13,28,43,58}$
7	0.900	$\beta_{7,22,37,52}$	7	0.875	$\beta_{14,29,44,59}$
8	1.000	0.000	8	1.000	0.000

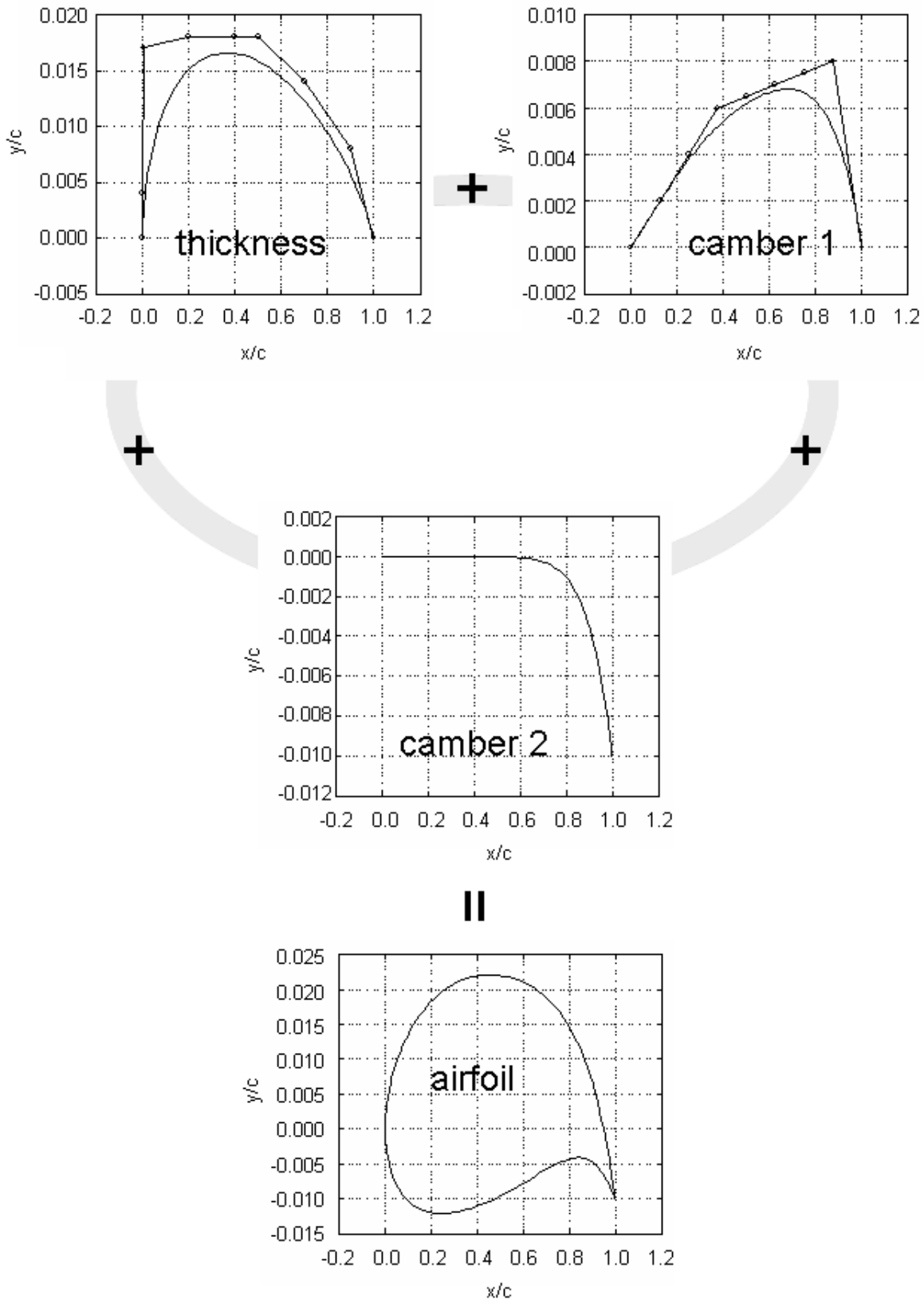


Figure 88: Method of generating an airfoil in each section.

Now, the optimization problem for the design of a wing at Mach 2.0 with an angle of attack of 0.0 [deg] can be defined as follows:

$$\begin{aligned}
\underset{\beta}{\text{Minimize}} \quad &: \quad C_D, && (67)\\
\text{Subject to} \quad &: \quad C_L \geq 0.1, && (68)\\
&\quad\; Wing\,volume \geq Wing\,volume_{initial}, && (69)\\
&\quad\; 0.03 \leq Airfoil\,thickness \leq 0.04, && (70)\\
&\quad\; -2.0 \leq Twist_D \leq Twist_C \leq Twist_B \leq Twist_A \leq 7.0, && (71)\\
&\quad\; Twist_B - Twist_C \leq 1.0. && (72)
\end{aligned}$$

Here, "wing volume" is non-dimensional volume when the chord length at Section A is set to one. Constraint equation 78 is imposed so that the twist angles between sections B and C do not change dramatically.

5.3.2 Validations of the Software

Validations for the flow and the derivative solvers are conducted below. At the end of this subsection, their required CPU times and memory sizes are summarized in Table 31.

5.3.2.1 The flow solver

The flow solver used in this optimization problem is three-dimensional Navier-Stokes/Euler code LANS developed by Fujii [34]. In this study, the Euler mode is selected due to the limitation of computer resources. In LANS, an implicit LU-ADI scheme [62] is used for time integration, and the Roe scheme [69], coupled with the third-order MUSCL approach [79], is used for the flux evaluation. Since LANS reads structured grids, the coarse grid shown in Figure 89 is treated as a low-fidelity model, and the

fine grid shown in Figure 90 is treated as a high-fidelity model. Grid topologies for both models are C-H, and since a symmetry condition is assumed in the computation, only the flow around the half wing is solved. The low-fidelity model has 33,201 nodes (51 in the chord-wise direction, 31 in the circumferential direction, and 21 from the body to the outer boundary) while the high-fidelity model has 159,681 (101 in the chord-wise direction, 51 in the circumferential direction, and 31 from the body to the outer boundary). The grid generator is an in-house software using transfinite interpolation [27], and grid orthogonality is enhanced near the body.

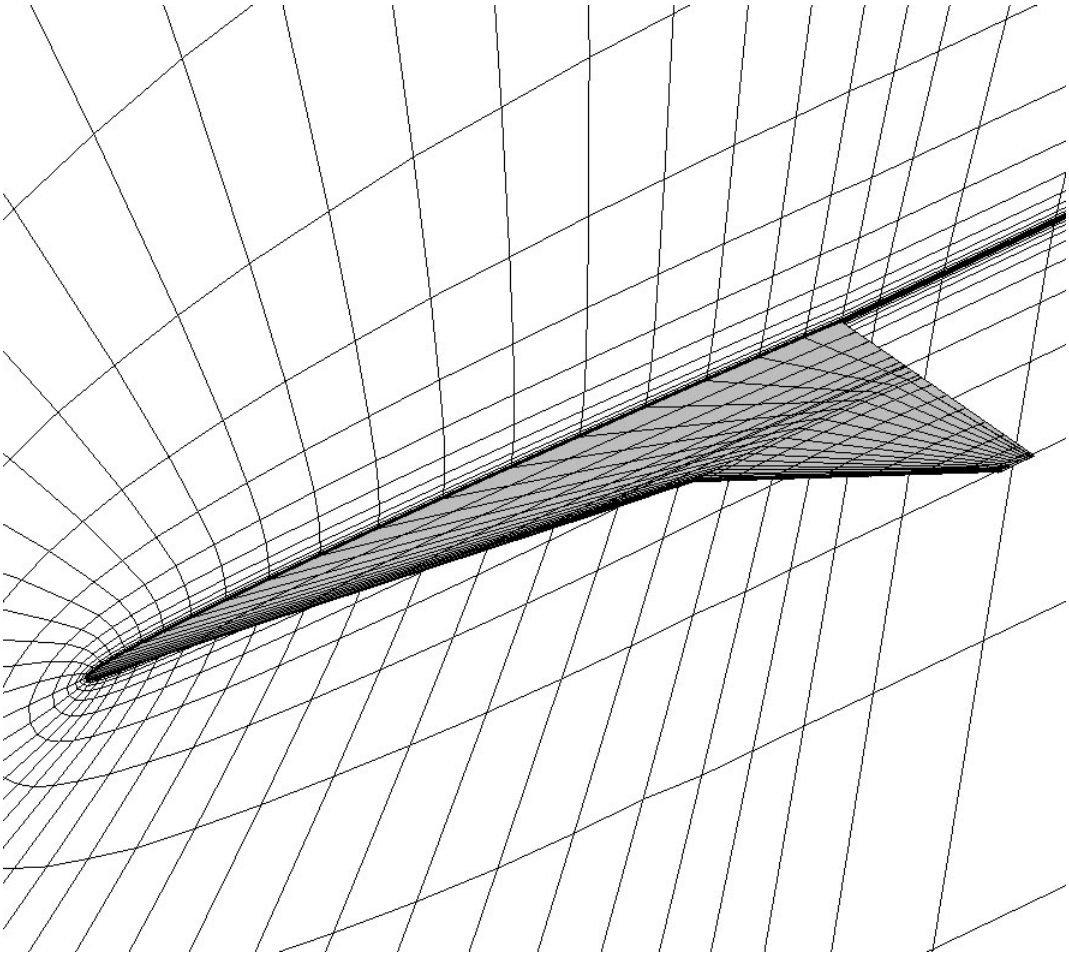


Figure 89: Coarse grid around the wing (51 × 31 × 21), low-fidelity model.

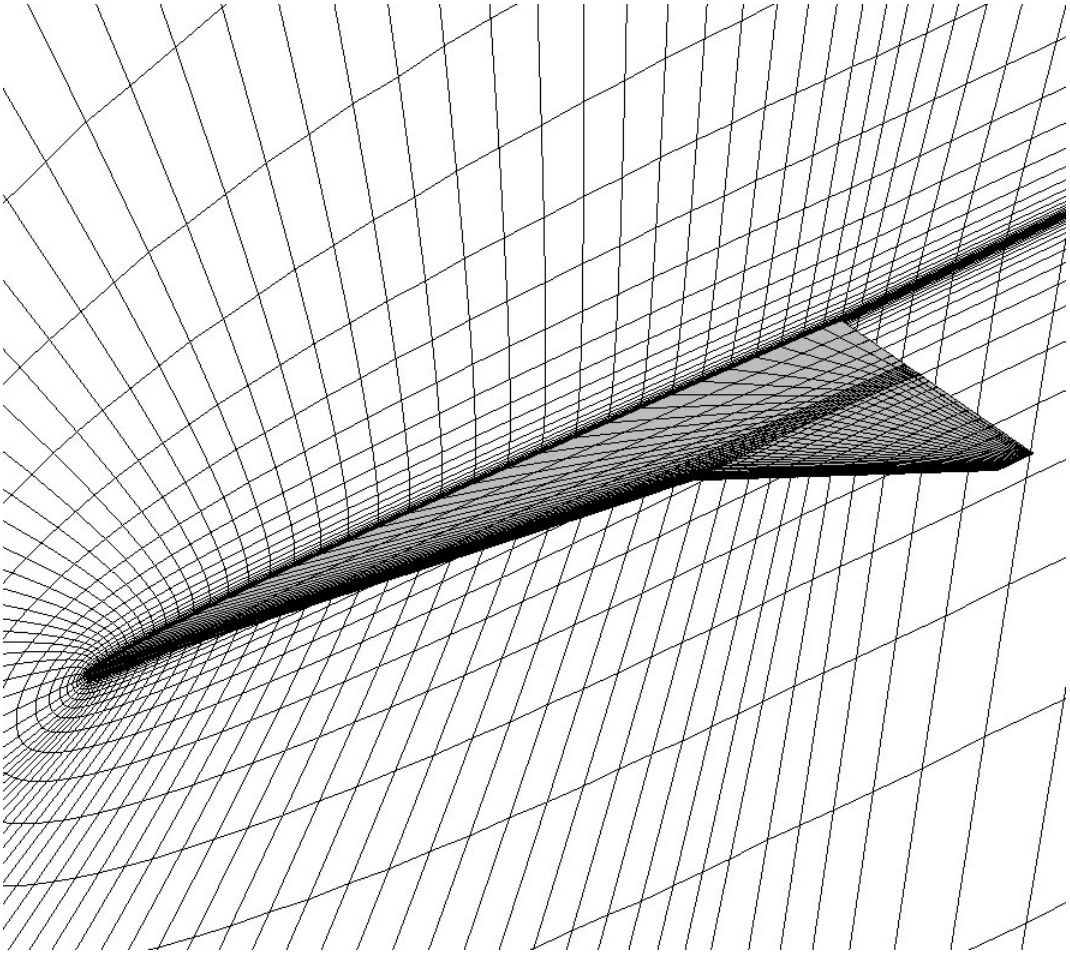


Figure 90: Fine grid around the wing (101 × 51 × 31), high-fidelity model.

Convergence histories in aerodynamics are shown in Figure 91 for the low-fidelity case, and in Figure 92, for the high-fidelity case. In both cases, the local-time stepping algorithm [52] is used in order to obtain converged solutions in a shorter time period. Since a normalized residual does not decrease well in this case, the history of aerodynamic coefficients is used as a determinant of convergence. Because LANS conducts a preprocessing computation in 30 iterations to enhance the stability of the computation, C_L and C_D remain at almost zero for 30 iterations in Figures 91 and 92. Judging from Figures 91 and 92, 100 iterations are implemented for the low-fidelity case, and 300 for the high-fidelity case in the entire optimization process.

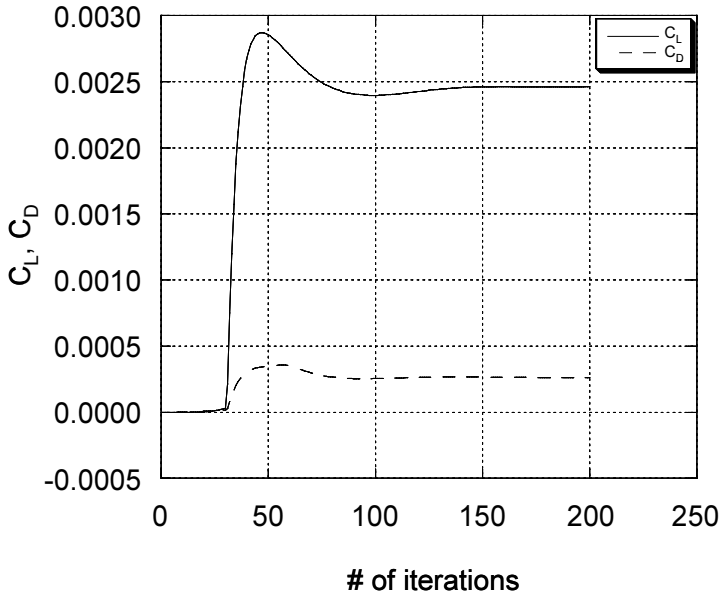


Figure 91: History of aerodynamic coefficients with the low-fidelity model.

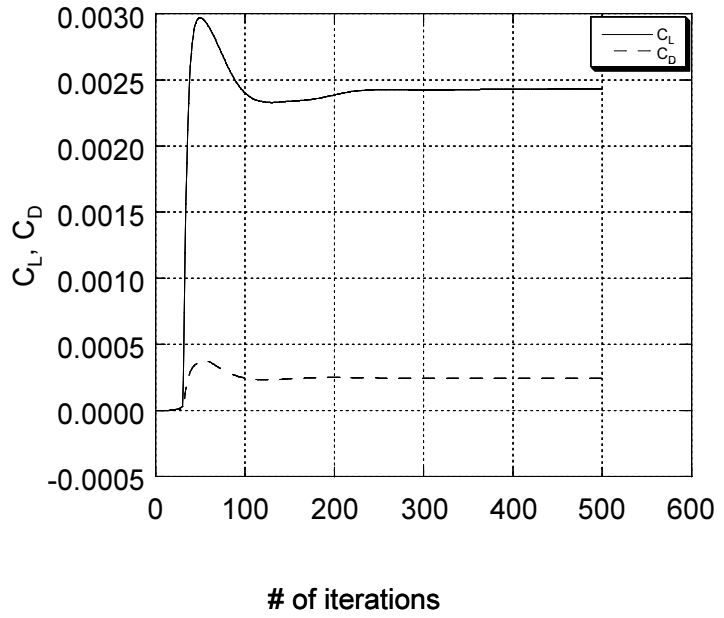


Figure 92: History of aerodynamic coefficients with the high-fidelity model.

Experimental data of the wing-body configuration shown in Figure 93, whose wing shape is depicted in Figure 87, and whose fuselage is an axisymmetric slender body, is available in Reference [44]. Since only the wing is analyzed in computer simulations and since a viscous term is neglected in computer simulations, only the computed lift can be compared with that of the experimental data because one can assume that the existence of either the fuselage or viscosity in the experimental data have little effect on lift.

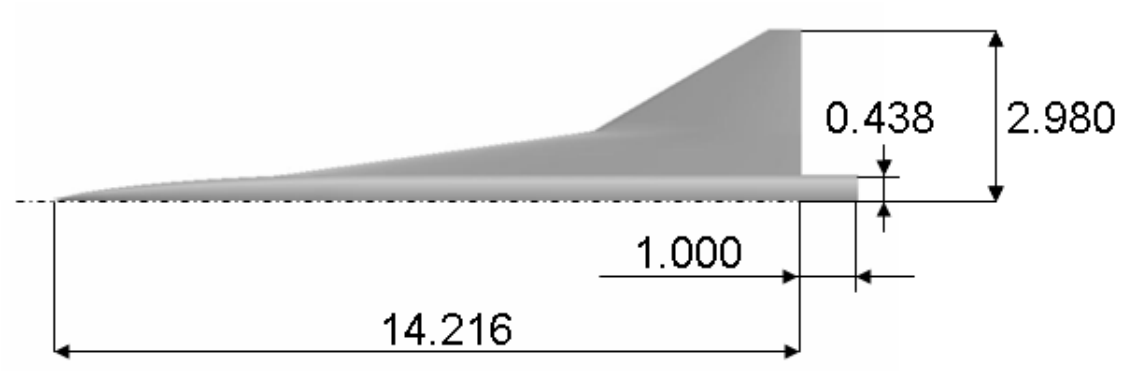


Figure 93: Model 9 in NASA-TN-D-4211 [44].

Figure 94 shows comparisons of the lift coefficients of the low-fidelity model, the high-fidelity model, PANAIR [71], and the experiment at Mach 1.98. Results with PANAIR are superimposed for the purpose of reference here. CFD results are slightly higher than the experimental results at a higher angle of attack, perhaps because of the flow separation in the experimental model at a higher angle of attack. Of the various CFD models, LANS and PANAIR compute almost identical lift coefficients.

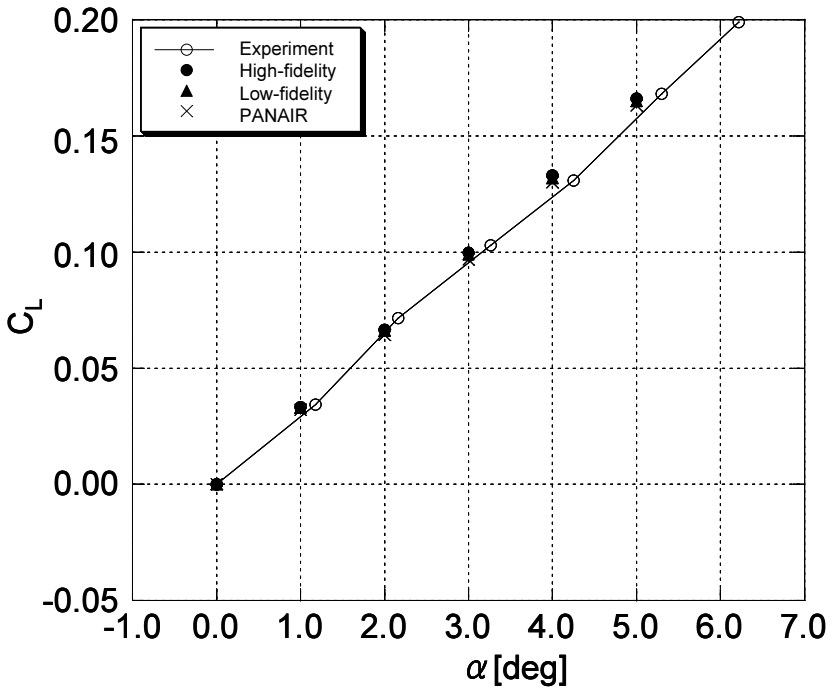


Figure 94: Comparisons of the lift coefficient.

Figure 95 shows comparisons of the drag coefficients of the low-fidelity model, the high-fidelity model, and PANAIR at Mach 1.98. Again, the results of PANAIR are superimposed for the purpose of reference here. One can see that the low-fidelity model computes slightly higher drag than the high-fidelity model. PANAIR computes a lower drag than LANS, but the orders of the values and the tendencies of the three models are the same. Since numerical accuracy is not the main focus of this study, this result is accepted as is.

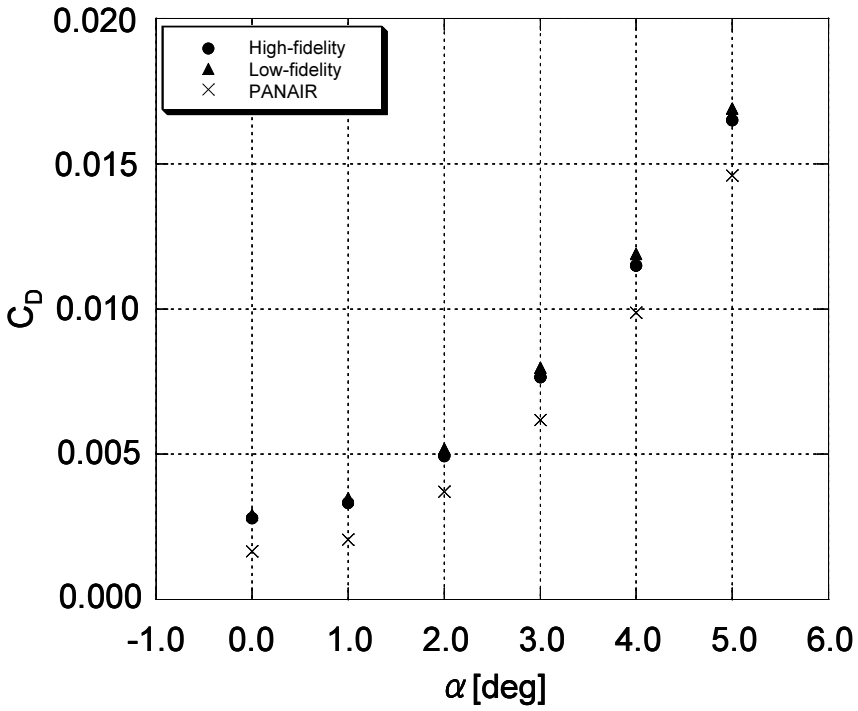


Figure 95: Comparisons of the drag coefficient.

5.3.2.2 The derivative solver

In derivative-based optimizations, obtaining accurate derivatives in a shorter time period is very important. The method proposed in Section 4.2 is implemented to calculate derivatives. For computing the derivatives of nodes with respect to shape parameters, TAPENADE is directly applied to the grid generator code because a structured grid is used in this study. For computing the derivatives of aerodynamics with respect to nodes, TAPENADE is applied to LANS, and the generated derivative code is modified so that it does not require such a huge memory size.

Tables 27 and 28 show the comparisons of the derivatives of the AD and the FD with the low-fidelity model. In the FD method, step sizes $d\beta_i (i = 1, ..., 64)$ are set to 0.01, 0.001, and 0.0001. In Tables 27 and 28, one can identify several numerical discrepancies between the results computed with the AD and those computed with the FD. However, as points $(\frac{\partial C_L}{\partial \beta_i}, \frac{\partial C_D}{\partial \beta_i})(i = 1, ..., 64)$, computed with the AD and the

FD, are superimposed in the same graphs in order to check the matching between the AD and FD (Figures 96, 97, 98, 99, and 100), one can confirm that the AD and the FD with the appropriate step size compute similar derivatives, although the FD with some improper step sizes are probably bothered by numerical noise. Here, the AD takes 219 seconds to compute all the derivatives while the FD takes 814 seconds for each step size. This finding indicates that the derivative code works accurately and that the AD computes derivatives more accurately and more rapidly than the FD.

Table 27: Derivatives computed by the AD and the FD with the low-fidelity model(1) (wing, M=2.0, α=1.0 [deg]).

	$\frac{\partial C_L}{\partial \beta_i}$				$\frac{\partial C_D}{\partial \beta_i}$			
i	AD	FD ($d\beta_i = 0.01$)	FD ($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)	AD	FD ($d\beta_i = 0.01$)	FD FD($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)
1	0.0057	0.0145	0.0135	0.0175	0.0799	0.0834	0.0684	0.0692
2	0.0036	0.0057	0.0070	-0.0179	0.0343	0.0339	0.0306	0.0324
3	-0.0021	-0.0054	-0.0067	-0.0135	0.0155	0.0164	0.0151	0.0165
4	-0.0095	-0.0137	-0.0115	-0.0105	0.0116	0.0134	0.0121	0.0146
5	-0.0150	-0.0162	-0.0162	-0.0168	0.0130	0.0151	0.0133	0.0176
6	-0.0153	-0.0137	-0.0145	-0.0174	0.0141	0.0169	0.0141	0.0178
7	-0.0091	-0.0078	-0.0065	-0.0259	0.0098	0.0187	0.0107	0.0097
8	0.1397	0.1790	0.1785	0.1549	0.0029	0.0099	0.0046	0.0029
9	0.1635	0.1792	0.1779	0.1681	0.0031	0.0063	0.0041	0.0040
10	0.1755	0.1692	0.1690	0.1657	0.0035	0.0053	0.0039	0.0032
11	0.1630	0.1462	0.1466	0.1627	0.0032	0.0046	0.0032	0.0069
12	0.1231	0.1095	0.1095	0.1095	0.0023	0.0042	0.0024	0.0066
13	0.0559	0.0540	0.0535	0.0440	0.0010	0.0047	0.0012	0.0048
14	-0.0492	-0.0394	-0.0424	-0.0337	-0.0008	0.0094	0.0006	0.0010
15	-0.8628	-0.8507	-0.8510	-0.8700	-0.0230	0.0004	-0.0205	-0.0225
16	0.0089	0.0117	0.0115	-0.0041	0.0167	0.0153	0.0150	0.0153
17	0.0039	0.0038	0.0064	0.0237	0.0095	0.0089	0.0083	0.0078
18	-0.0010	-0.0027	-0.0011	0.0210	0.0058	0.0058	0.0056	0.0019
19	-0.0043	-0.0062	-0.0063	0.0047	0.0055	0.0060	0.0054	0.0028
20	-0.0055	-0.0064	-0.0032	0.0239	0.0067	0.0073	0.0067	0.0073
21	-0.0048	-0.0053	-0.0066	0.0210	0.0072	0.0080	0.0071	0.0052
22	-0.0029	-0.0025	-0.0038	0.0078	0.0051	0.0078	0.0051	0.0060
23	0.1153	0.1112	0.1137	0.1182	0.0027	0.0042	0.0032	0.0005
24	0.0953	0.0919	0.0993	0.1106	0.0018	0.0028	0.0022	-0.0012
25	0.0782	0.0732	0.0759	0.0791	0.0015	0.0020	0.0019	0.0022
26	0.0594	0.0565	0.0609	0.0706	0.0010	0.0015	0.0014	0.0012
27	0.0384	0.0374	0.0387	0.0446	0.0006	0.0012	0.0006	-0.0025
28	0.0137	0.0163	0.0127	0.0221	0.0001	0.0012	-0.0001	-0.0010
29	-0.0258	-0.0204	-0.0189	-0.0062	-0.0006	0.0025	-0.0005	-0.0025
30	-0.4338	-0.4303	-0.4351	-0.4459	-0.0120	-0.0048	-0.0113	-0.0154

Table 28: Derivatives computed by the AD and the FD with the low-fidelity model(2) (wing, M=2.0, α=1.0 [deg]).

	$\frac{\partial C_L}{\partial \beta_i}$				$\frac{\partial C_D}{\partial \beta_i}$			
i	AD	FD ($d\beta_i = 0.01$)	FD ($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)	AD	FD ($d\beta_i = 0.01$)	FD FD($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)
31	0.0137	-0.0013	-0.0004	-0.0352	0.0178	0.0393	0.0220	0.0191
32	-0.0158	-0.0157	-0.0183	-0.0459	0.0126	0.0200	0.0149	0.0179
33	-0.0188	-0.0142	-0.0176	0.0005	0.0155	0.0193	0.0167	0.0184
34	-0.0113	-0.0085	-0.0153	0.0196	0.0178	0.0200	0.0182	0.0193
35	-0.0046	-0.0042	-0.0095	-0.0375	0.0179	0.0198	0.0184	0.0183
36	-0.0012	-0.0017	-0.0005	-0.0159	0.0159	0.0184	0.0165	0.0169
37	0.0010	0.0004	0.0003	-0.0610	0.0104	0.0163	0.0111	0.0118
38	0.1628	0.1312	0.1280	0.1362	0.0037	0.0125	0.0028	0.0035
39	0.0521	0.0460	0.0370	0.0636	0.0001	0.0036	0.0000	0.0019
40	0.0096	0.0144	0.0102	0.0115	-0.0012	0.0015	-0.0006	-0.0006
41	-0.0059	0.0002	0.0031	0.0035	-0.0014	0.0008	-0.0006	-0.0013
42	-0.0117	-0.0065	-0.0070	-0.0116	-0.0013	0.0009	-0.0010	0.0006
43	-0.0189	-0.0150	-0.0188	-0.0133	-0.0012	0.0016	-0.0006	0.0010
44	-0.0458	-0.0438	-0.0459	-0.0476	-0.0014	0.0047	-0.0008	0.0008
45	-0.4797	-0.4805	-0.4768	-0.4729	-0.0159	-0.0002	-0.0139	-0.0119
46	0.0053	-0.0006	0.0044	-0.0235	0.0029	0.0040	0.0028	0.0018
47	-0.0003	-0.0021	-0.0024	-0.0281	0.0022	0.0026	0.0023	0.0016
48	-0.0020	-0.0019	-0.0025	-0.0300	0.0026	0.0030	0.0028	0.0024
49	-0.0015	-0.0011	0.0008	-0.0425	0.0031	0.0032	0.0031	0.0024
50	-0.0006	-0.0003	-0.0036	-0.0028	0.0031	0.0032	0.0030	0.0032
51	-0.0001	-0.0001	-0.0011	0.0267	0.0027	0.0029	0.0028	0.0027
52	0.0003	0.0003	0.0001	-0.0245	0.0018	0.0022	0.0019	0.0023
53	0.0223	0.0172	0.0183	0.0424	0.0005	0.0009	0.0003	-0.0002
54	0.0047	0.0056	0.0004	0.0006	0.0000	0.0001	-0.0002	-0.0008
55	0.0000	0.0005	0.0014	-0.0106	-0.0002	-0.0001	-0.0002	-0.0008
56	-0.0015	-0.0004	-0.0018	-0.0271	-0.0002	-0.0001	-0.0002	-0.0017
57	-0.0020	-0.0010	-0.0013	-0.0096	-0.0002	-0.0001	-0.0002	-0.0001
58	-0.0030	-0.0026	-0.0022	-0.0061	-0.0002	0.0000	-0.0002	0.0000
59	-0.0069	-0.0072	-0.0083	-0.0457	-0.0002	0.0001	-0.0003	-0.0008
60	-0.0711	-0.0724	-0.0755	-0.0793	-0.0026	-0.0016	-0.0026	-0.0034
61	0.0105	0.0105	0.0102	0.0043	0.0003	0.0004	0.0006	0.0026
62	0.0060	0.0061	0.0056	0.0245	0.0002	0.0002	0.0002	-0.0016
63	0.0094	0.0094	0.0038	0.0298	0.0004	0.0004	0.0006	0.0024
64	0.0013	0.0014	0.0025	0.0154	0.0001	0.0001	0.0001	0.0007

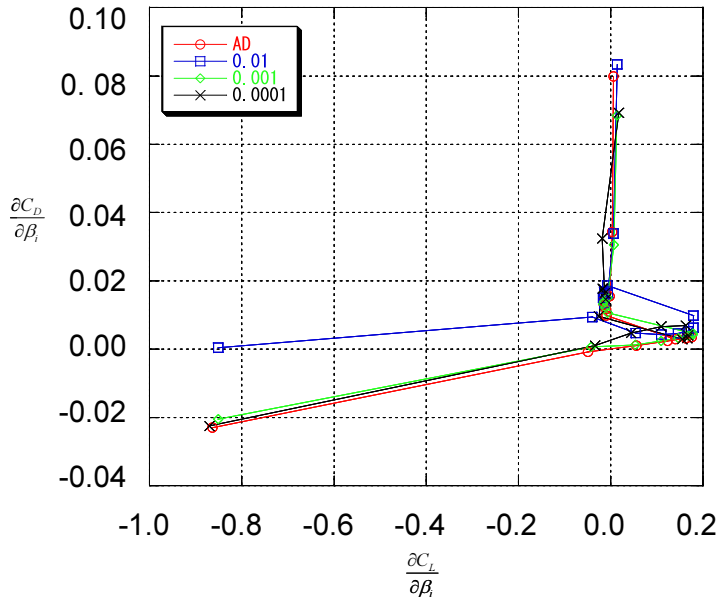


Figure 96: Comparison of the derivatives of the AD and the FD with the low-fidelity model (Section A, M=2.0, α=1.0 [deg]).

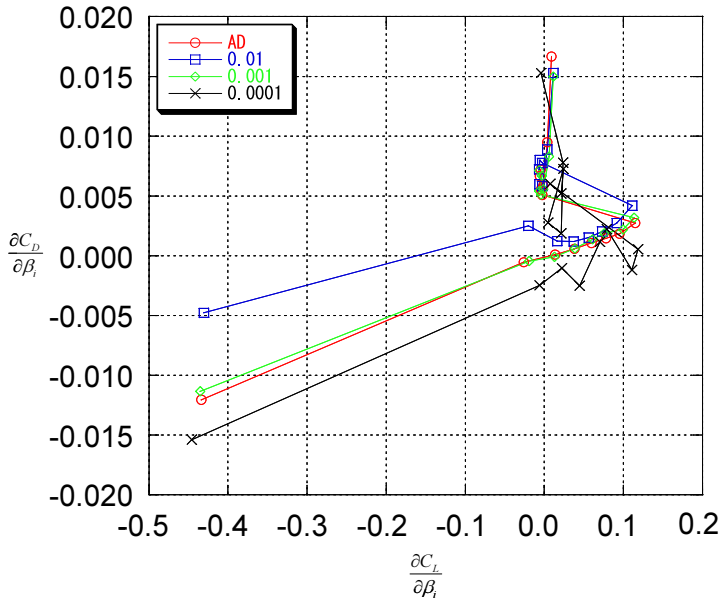


Figure 97: Comparison of the derivatives of the AD and the FD with the low-fidelity model (Section B, M=2.0, α=1.0 [deg]).

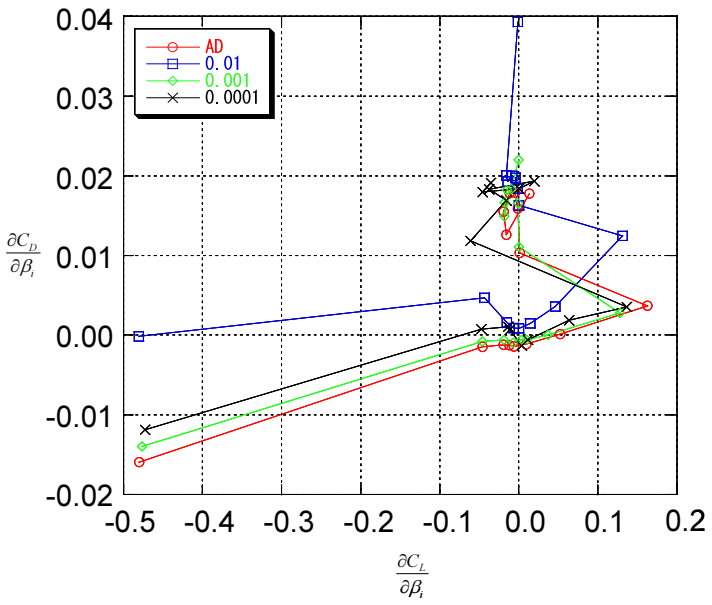


Figure 98: Comparison of the derivatives of the AD and the FD with the low-fidelity model (Section C, M=2.0, α=1.0 [deg]).

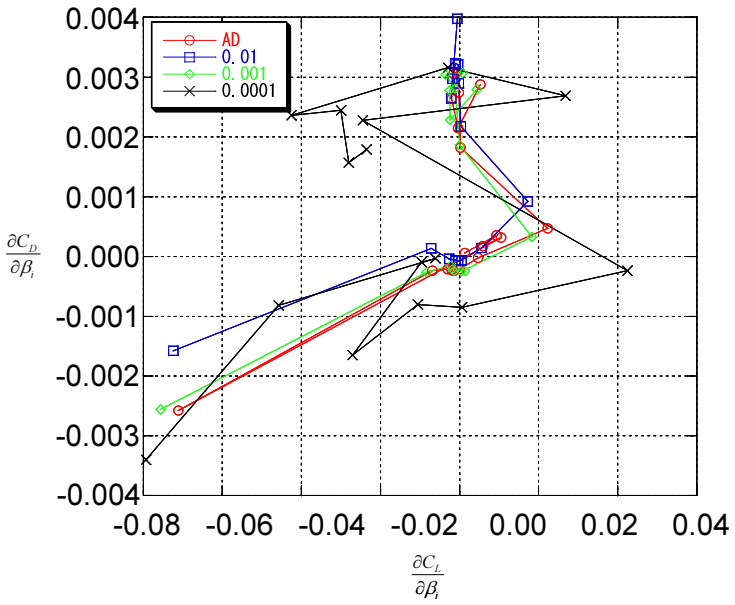


Figure 99: Comparison of the derivatives of the AD and the FD with the low-fidelity model (Section D, M=2.0, α=1.0 [deg]).

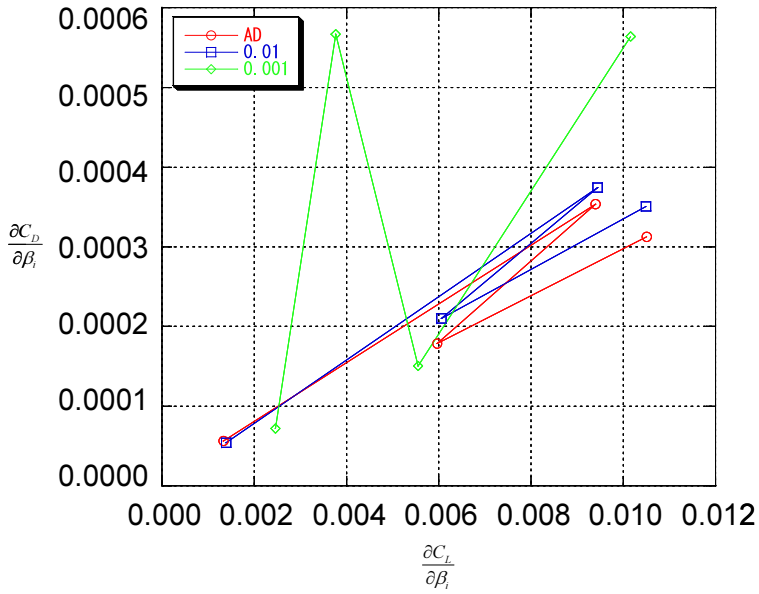


Figure 100: Comparison of the derivatives of the AD and the FD with the low-fidelity model (twist angles, M=2.0, α=1.0 [deg]).

The derivatives of the AD and the FD with the high-fidelity model are also compared. In the FD method, step sizes $d\beta_i (i = 1, ..., 64)$ are set to 0.01, 0.001, and 0.0001. Tables 29 and 30 and Figures 101, 102, 103, 104, and 105 summarize the results. Here, the AD takes 3,197 seconds while the FD takes 14,791 seconds in order to compute all the derivatives. Again, one can conclude that the AD works well and that the AD computes derivatives more accurately and more rapidly than the FD.

Table 29: Derivatives computed by the AD and the FD with the high-fidelity model(1) (wing, M=2.0, α=1.0 [deg]).

	$\frac{\partial C_L}{\partial \beta_i}$				$\frac{\partial C_D}{\partial \beta_i}$			
i	AD	FD ($d\beta_i = 0.01$)	FD ($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)	AD	FD ($d\beta_i = 0.01$)	FD FD($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)
1	0.0068	0.0059	0.0059	0.0090	0.0809	0.0779	0.0809	0.0782
2	0.0044	0.0042	0.0042	0.0081	0.0317	0.0313	0.0317	0.0317
3	0.0004	0.0004	0.0004	0.0048	0.0146	0.0144	0.0146	0.0144
4	-0.0075	-0.0074	-0.0074	-0.0047	0.0119	0.0118	0.0119	0.0116
5	-0.0160	-0.0159	-0.0159	-0.0156	0.0145	0.0143	0.0145	0.0142
6	-0.0181	-0.0180	-0.0180	-0.0131	0.0168	0.0165	0.0168	0.0169
7	-0.0116	-0.0106	-0.0106	-0.0135	0.0147	0.0134	0.0147	0.0134
8	0.1280	0.1306	0.1306	0.1294	0.0031	0.0025	0.0031	0.0025
9	0.1721	0.1712	0.1712	0.1750	0.0036	0.0033	0.0036	0.0035
10	0.2043	0.2021	0.2021	0.2022	0.0044	0.0043	0.0044	0.0042
11	0.1972	0.1953	0.1953	0.1979	0.0045	0.0043	0.0045	0.0044
12	0.1443	0.1430	0.1430	0.1462	0.0034	0.0032	0.0034	0.0032
13	0.0625	0.0620	0.0620	0.0684	0.0019	0.0015	0.0019	0.0015
14	-0.0384	-0.0375	-0.0375	-0.0311	0.0010	-0.0003	0.0010	0.0000
15	-0.9278	-0.9275	-0.9275	-0.9235	-0.0216	-0.0244	-0.0216	-0.0244
16	0.0128	0.0095	0.0095	0.0148	0.0141	0.0140	0.0141	0.0138
17	0.0063	0.0056	0.0056	0.0183	0.0074	0.0072	0.0074	0.0070
18	0.0005	0.0002	0.0002	-0.0059	0.0047	0.0046	0.0047	0.0053
19	-0.0045	-0.0055	-0.0055	-0.0035	0.0052	0.0052	0.0052	0.0051
20	-0.0069	-0.0068	-0.0068	-0.0062	0.0071	0.0071	0.0071	0.0072
21	-0.0063	-0.0056	-0.0056	-0.0080	0.0084	0.0083	0.0084	0.0077
22	-0.0039	-0.0042	-0.0042	0.0020	0.0074	0.0070	0.0074	0.0072
23	0.1072	0.1073	0.1073	0.1027	0.0025	0.0022	0.0025	0.0027
24	0.1018	0.1014	0.1014	0.0983	0.0022	0.0021	0.0022	0.0016
25	0.0912	0.0906	0.0906	0.0961	0.0021	0.0021	0.0021	0.0020
26	0.0700	0.0697	0.0697	0.0731	0.0016	0.0016	0.0016	0.0016
27	0.0428	0.0430	0.0430	0.0354	0.0010	0.0009	0.0010	0.0006
28	0.0159	0.0155	0.0155	0.0057	0.0004	0.0003	0.0004	0.0001
29	-0.0173	-0.0172	-0.0172	-0.0150	0.0000	-0.0003	0.0000	-0.0009
30	-0.4605	-0.4605	-0.4605	-0.4618	-0.0116	-0.0125	-0.0116	-0.0125

Table 30: Derivatives computed by the AD and the FD with the high-fidelity model(2) (wing, M=2.0, α=1.0 [deg]).

	$\frac{\partial C_L}{\partial \beta_i}$				$\frac{\partial C_D}{\partial \beta_i}$			
i	AD	FD ($d\beta_i = 0.01$)	FD ($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)	AD	FD ($d\beta_i = 0.01$)	FD FD($d\beta_i = 0.001$)	FD ($d\beta_i = 0.0001$)
31	0.0411	0.0358	0.0358	0.0173	0.0141	0.0101	0.0141	0.0113
32	-0.0113	-0.0092	-0.0092	-0.0356	0.0074	0.0072	0.0074	0.0065
33	-0.0247	-0.0229	-0.0229	-0.0541	0.0128	0.0129	0.0128	0.0121
34	-0.0169	-0.0212	-0.0212	-0.0273	0.0169	0.0169	0.0169	0.0168
35	-0.0072	-0.0105	-0.0105	-0.0226	0.0180	0.0179	0.0180	0.0178
36	-0.0021	-0.0021	-0.0021	-0.0495	0.0169	0.0166	0.0169	0.0158
37	-0.0002	-0.0035	-0.0035	-0.0431	0.0129	0.0122	0.0129	0.0112
38	0.1558	0.1548	0.1548	0.1449	0.0051	0.0042	0.0051	0.0039
39	0.0564	0.0564	0.0564	0.0544	0.0013	0.0009	0.0013	0.0005
40	0.0116	0.0126	0.0126	-0.0059	-0.0009	-0.0012	-0.0009	-0.0013
41	-0.0060	-0.0080	-0.0080	-0.0071	-0.0014	-0.0016	-0.0014	-0.0017
42	-0.0120	-0.0143	-0.0143	-0.0160	-0.0013	-0.0015	-0.0013	-0.0018
43	-0.0154	-0.0149	-0.0149	-0.0382	-0.0010	-0.0013	-0.0010	-0.0019
44	-0.0342	-0.0364	-0.0364	-0.0470	-0.0006	-0.0014	-0.0006	-0.0019
45	-0.5087	-0.5121	-0.5121	-0.5585	-0.0138	-0.0157	-0.0138	-0.0163
46	0.0054	0.0014	-0.0080	-0.0537	0.0030	0.0021	0.0023	0.0014
47	-0.0014	-0.0014	0.0232	-0.0521	0.0020	0.0017	0.0021	0.0010
48	-0.0029	0.0015	0.0221	-0.0219	0.0026	0.0024	0.0028	0.0022
49	-0.0021	0.0005	0.0193	0.0008	0.0031	0.0029	0.0034	0.0028
50	-0.0010	-0.0008	0.0213	0.2369	0.0032	0.0030	0.0034	0.0069
51	-0.0003	0.0000	0.0244	0.2371	0.0030	0.0028	0.0032	0.0070
52	0.0000	0.0043	0.0238	-0.3740	0.0026	0.0020	0.0025	-0.0041
53	0.0168	0.0226	0.0144	-0.0016	0.0009	0.0002	0.0003	-0.0001
54	0.0057	0.0087	0.0046	0.0085	0.0003	0.0000	0.0000	0.0002
55	0.0017	0.0061	-0.0345	-0.0071	0.0000	-0.0002	-0.0008	-0.0001
56	-0.0003	-0.0021	0.0231	-0.0336	-0.0002	-0.0003	0.0001	-0.0009
57	-0.0011	0.0016	-0.0085	0.4039	-0.0001	-0.0003	-0.0003	0.0071
58	-0.0017	0.0043	0.0294	0.4089	0.0001	-0.0002	0.0003	0.0069
59	-0.0046	-0.0013	-0.0054	0.3918	0.0003	-0.0002	-0.0002	0.0066
60	-0.0726	-0.0704	-0.0527	0.3587	-0.0014	-0.0025	-0.0021	0.0046
61	0.0103	0.0103	0.0110	0.0213	0.0003	0.0003	0.0003	0.0005
62	0.0059	0.0059	0.0058	0.0068	0.0002	0.0002	0.0002	0.0005
63	0.0095	0.0094	0.0063	-0.0051	0.0004	0.0003	0.0003	-0.0001
64	0.0013	0.0013	-0.0018	-0.0240	0.0001	0.0001	0.0000	-0.0003

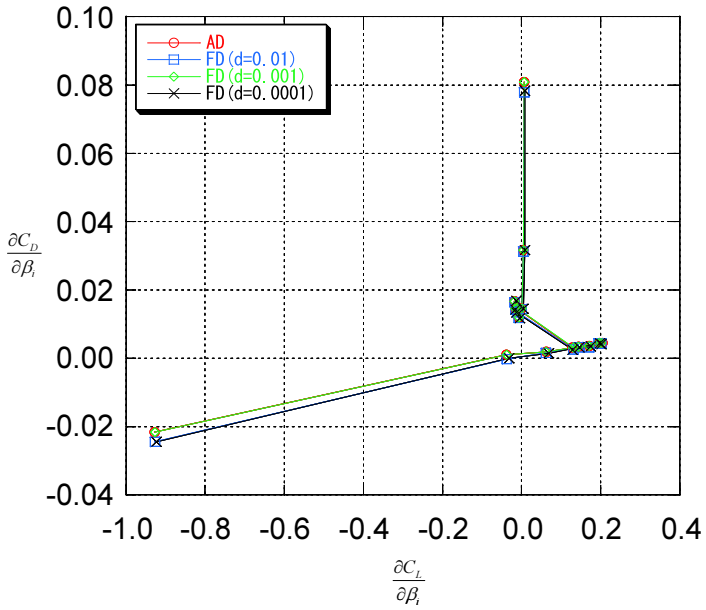


Figure 101: Comparison of the derivatives of the AD and the FD with the high-fidelity model (Section A, M=2.0, α=1.0 [deg]).

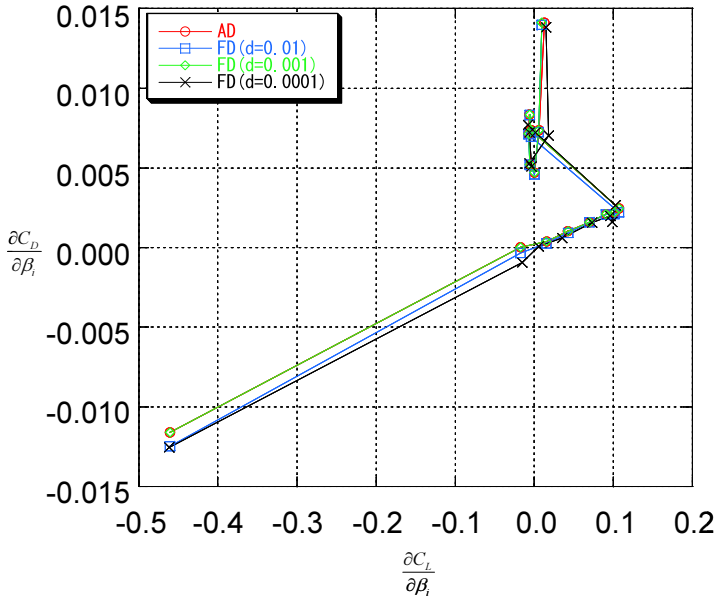


Figure 102: Comparison of the derivatives of the AD and the FD with the high-fidelity model (Section B, M=2.0, α=1.0 [deg]).

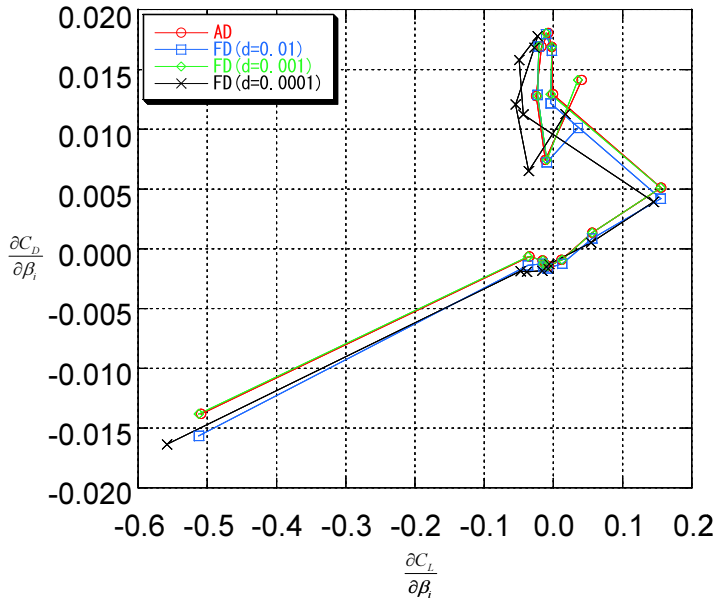


Figure 103: Comparison of the derivatives of the AD and the FD with the high-fidelity model (Section C, M=2.0, α=1.0 [deg]).

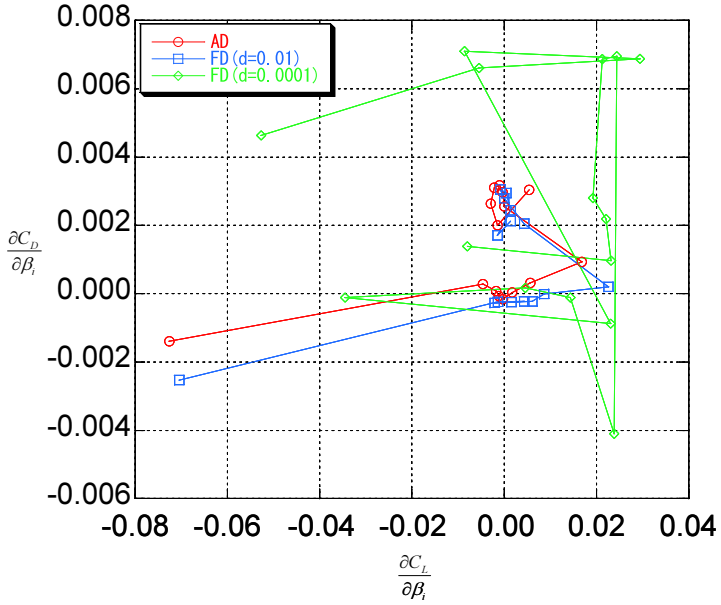


Figure 104: Comparison of the derivatives of the AD and the FD with the high-fidelity model (Section D, M=2.0, α=1.0 [deg]).

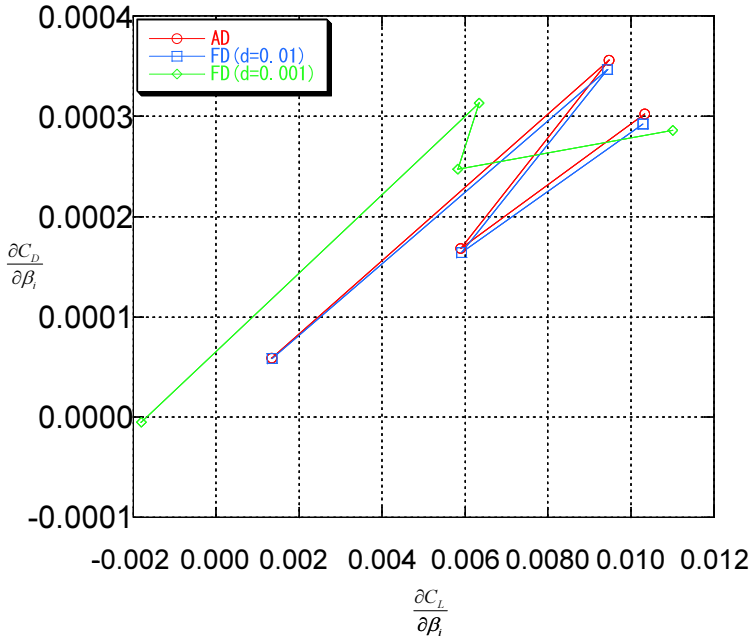


Figure 105: Comparison of the derivatives of the AD and the FD with the high-fidelity model (twist angles, M=2.0, α=1.0 [deg]).

As a summary of the validations of the software, the characteristics of the low- and high-fidelity models in the wing design problem are tabulated in Table 31, in which the values in parentheses indicate the number of iterations required.

Table 31: Characteristics of the low- and high-fidelity models (wing-optimization problem).

		Low fidelity $51 \times 31 \times 21 = 33,201$ nodes	High fidelity $2101 \times 51 \times 31 = 159,681$ nodes
Flow solver	CPU time [sec]	12.2 (100)	225.5 (300)
	Memory [MB]	25.8	59.6
Derivative code	CPU time [sec]	109.3 (100)	1,598.5 (300)
	Memory [MB]	119.5	311.6

5.3.3 Optimization with the Low-Fidelity Model

The wing is optimized with the low-fidelity model by means of the SQP in MATLAB. Here, the termination tolerance of the function value is set to 10^{-6}, that of the design variables 10^{-6}, and that of the constraint violation 10^{-3}. Default values are used for the remaining optimization parameters [2].

Figures 106, 107, and 108 show the histories of drag coefficients, lift coefficients, and volume, respectively. Here, volume is non-dimensional volume when the chord length at Section A is set to one. Initial and optimized aerodynamic coefficients, volume, and CPU time, and the number of function calls required in order to obtain an optimized wing are summarized in Table 32.

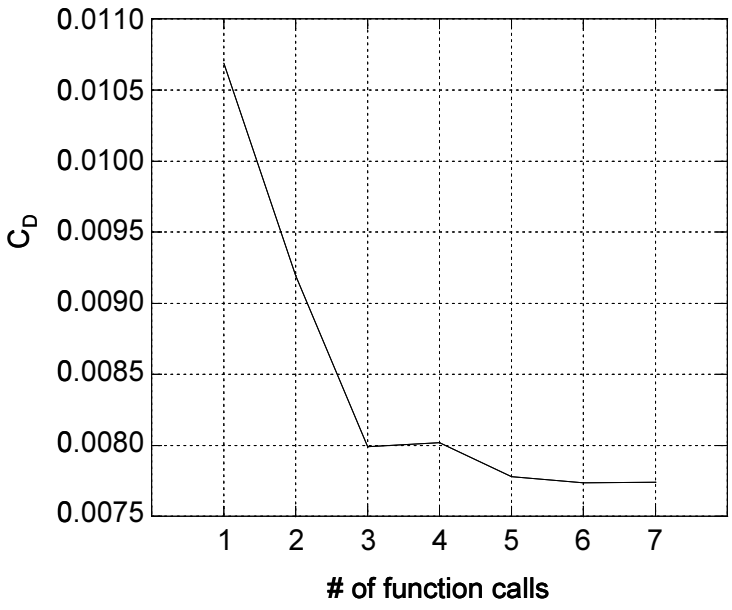


Figure 106: History of drag coefficients in the SQP with the low-fidelity model.

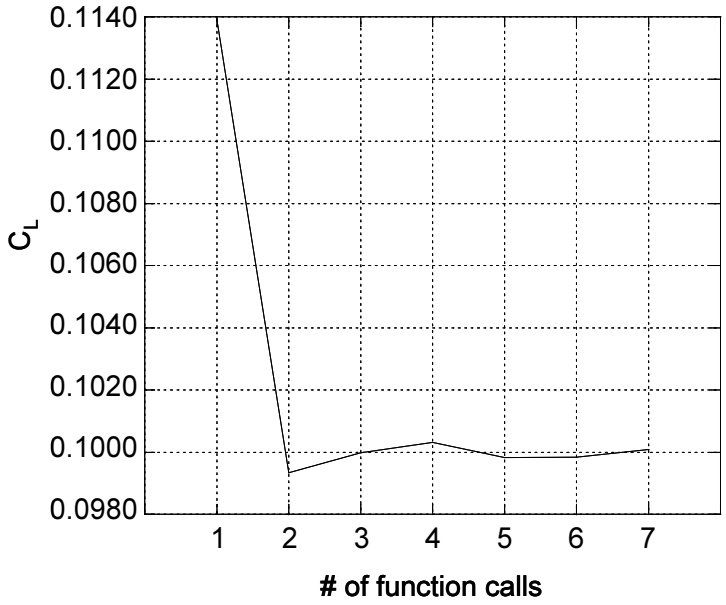


Figure 107: History of lift coefficients in the SQP with the low-fidelity model.

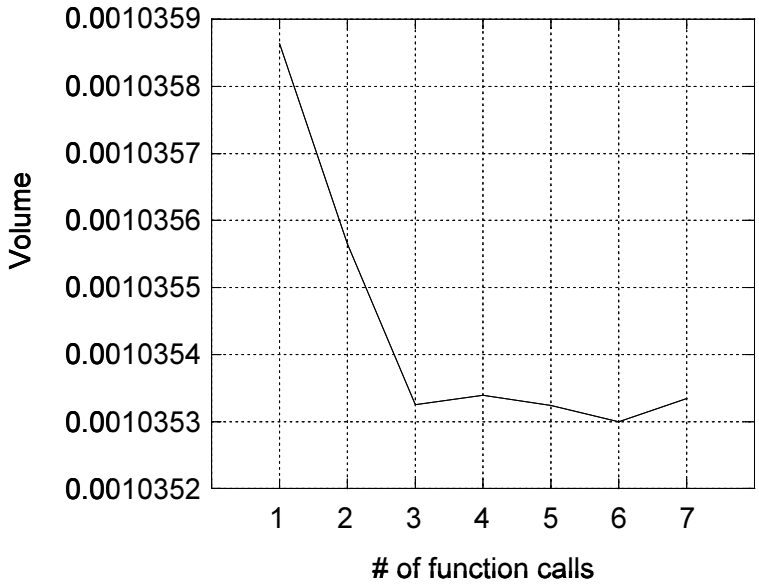


Figure 108: History of volume in the SQP with the low-fidelity model.

Table 32: Results with the low-fidelity model.

	C_D	C_L	Volume	No. of f_{low}, ∇f_{low} calls	CPU time [sec]
Initial	0.0107	0.1139	0.00104	1, 0	12
Optimized	0.0077	0.1001	0.00104	7, 7	1,665

The wing optimized with the low-fidelity model is analyzed with the high-fidelity model. In this case, C_D is 0.0075 and C_L is 0.1024, requiring an additional 226 seconds. Although the constraint of the lift coefficient ($C_L > 0.1$) is satisfied, this result is coincidental because the constraints may be violated in different optimization problems if the wing is optimized only with the low-fidelity model and analyzed with the high-fidelity model at the end.

Figure 109 shows the initial and optimized airfoils at sections A, B, C, and D. Since the scales of the horizontal and vertical axes differ in Figures 109(a), 109(b), 109(c), and 109(d), one should note that the angles of attack shown in these figures differ from the true angles of attack. Figure 110 shows the C_p distributions around the initial and optimized airfoils at sections A, B, C, and D, which are computed with the high-fidelity model. Figures 111 and 112 show a top/bottom view and a front view of C_p contours on the surfaces of the initial and optimized wings, respectively, visualized with FieldView [1].

By comparing the initial and optimized wings, one can observe the following phenomena:

- As Figures 111 and 112 clearly illustrate, the high pressure area on the leading edge from Sections C to D shrinks in the optimized wing because the lower leading edge from sections C to D in the optimized wing is compressed upward and becomes flatter, which eventually mitigates the compression of the shock

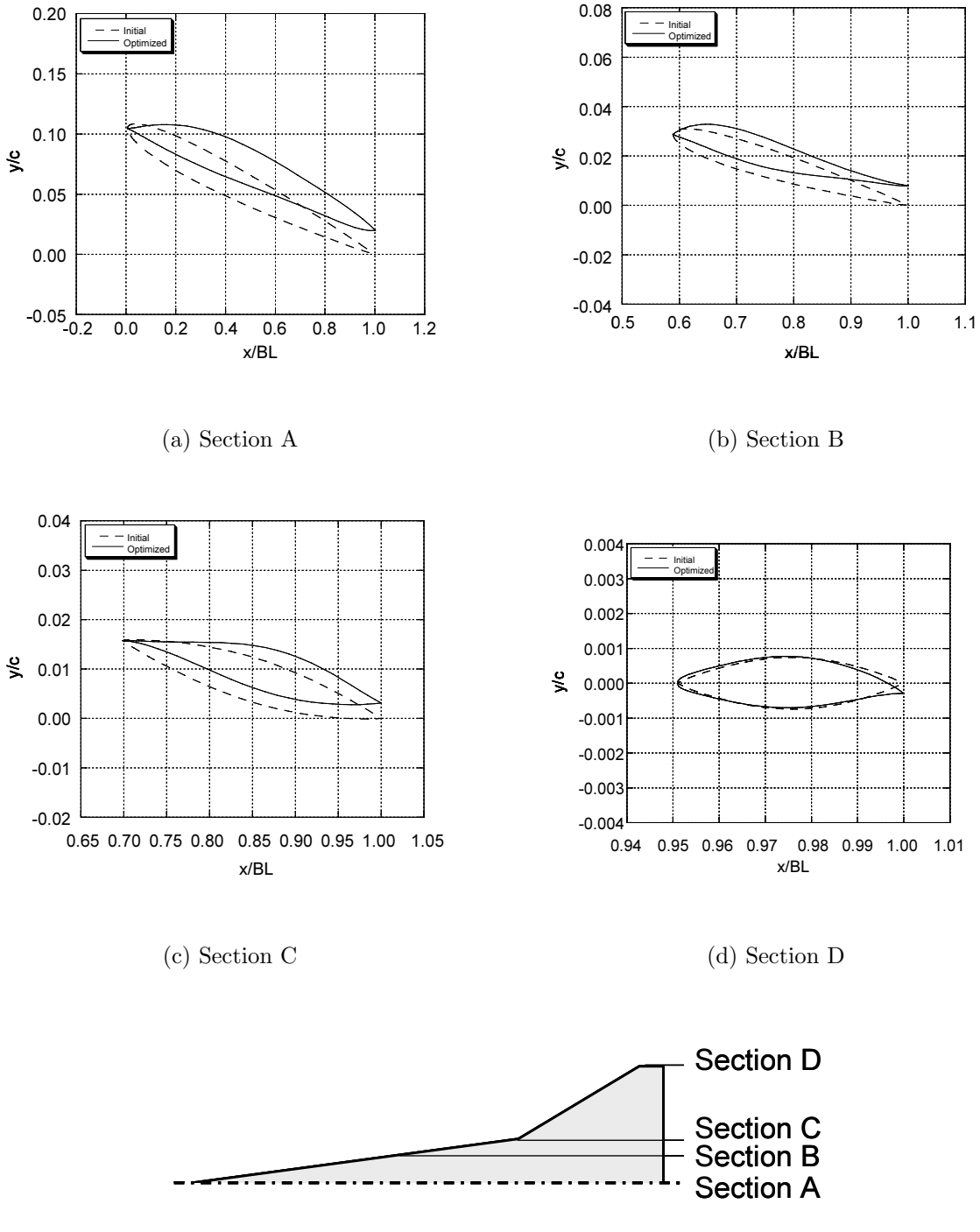
wave.

- As Figure 111 clearly shows, the high pressure area on the nose of the initial wing almost diminishes in the optimized wing because of the flat lower leading edge around the nose in the optimized wing (Figure 109(a)), which eventually mitigates the compression of the shock wave.

- As Figure 111 clearly shows, the projected frontal area shrinks in the optimized wing because of the optimized twist angles in each section.

- All of above phenomena reduce drag.

- In order to compensate for the volume reduced with the flattened lower surface of the wing, the upper surface of the optimized wing has a more rounded curvature, shown in Figure 109, mitigating the magnitude of expansion on the upper surface of the wing, depicted in Figures 110 and 111, and eventually losing partial lift. However, since the initial wing has enough lift, the optimized wing still satisfies the constraints pertaining to lift.



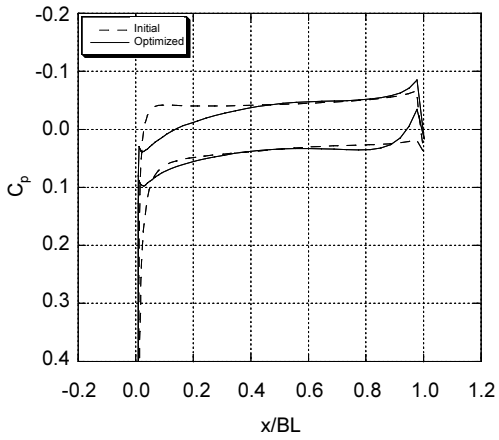
(a) Section A

(b) Section B

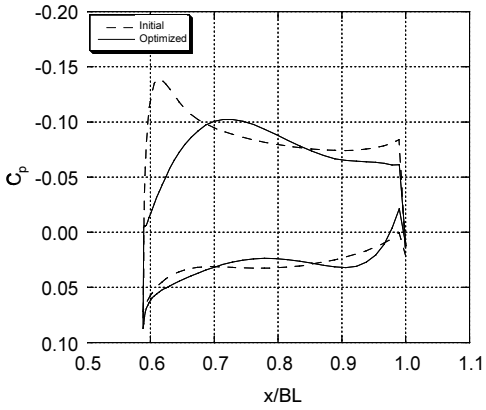
(c) Section C

(d) Section D

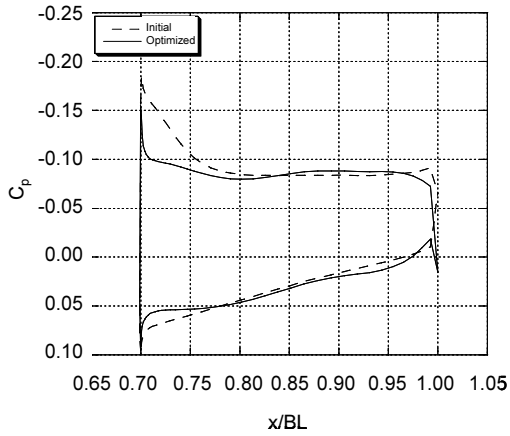
Figure 109: Shapes of the initial and optimized wing sections (optimized through the SQP with the low-fidelity model).



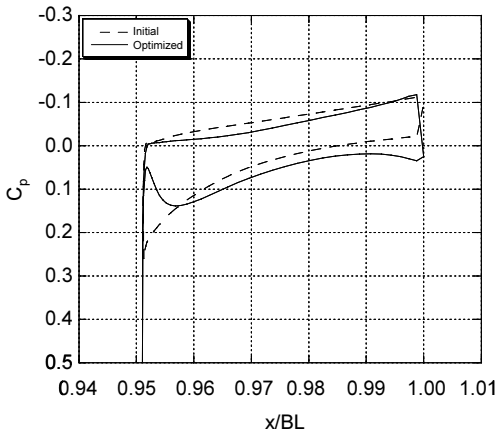
(a) Section A



(b) Section B



(c) Section C



(d) Section D

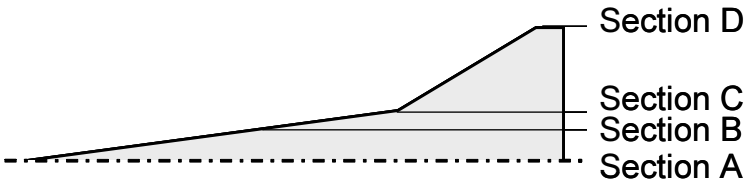
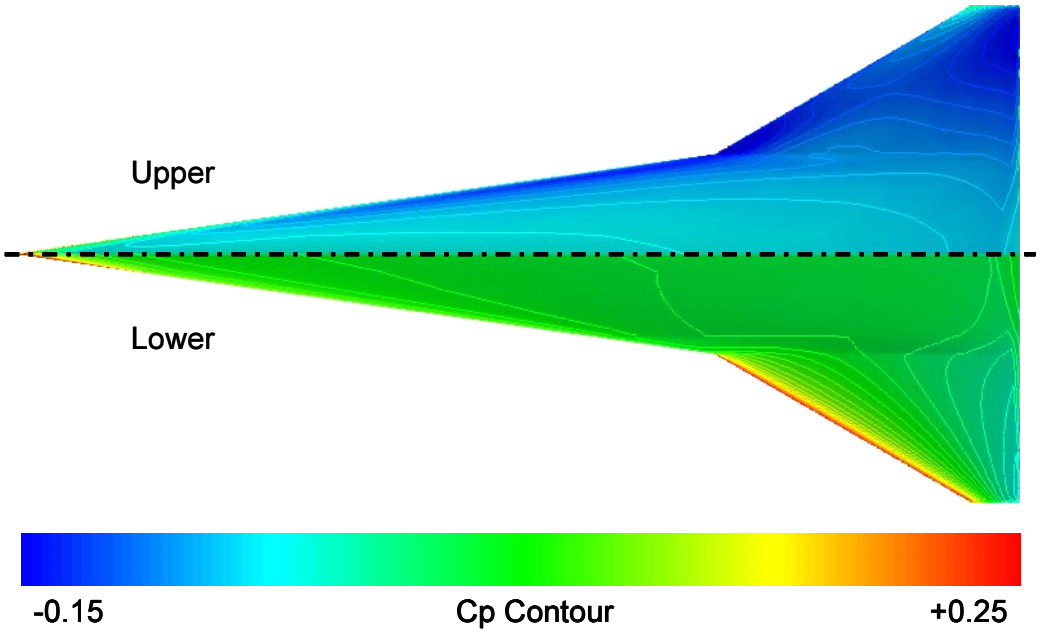
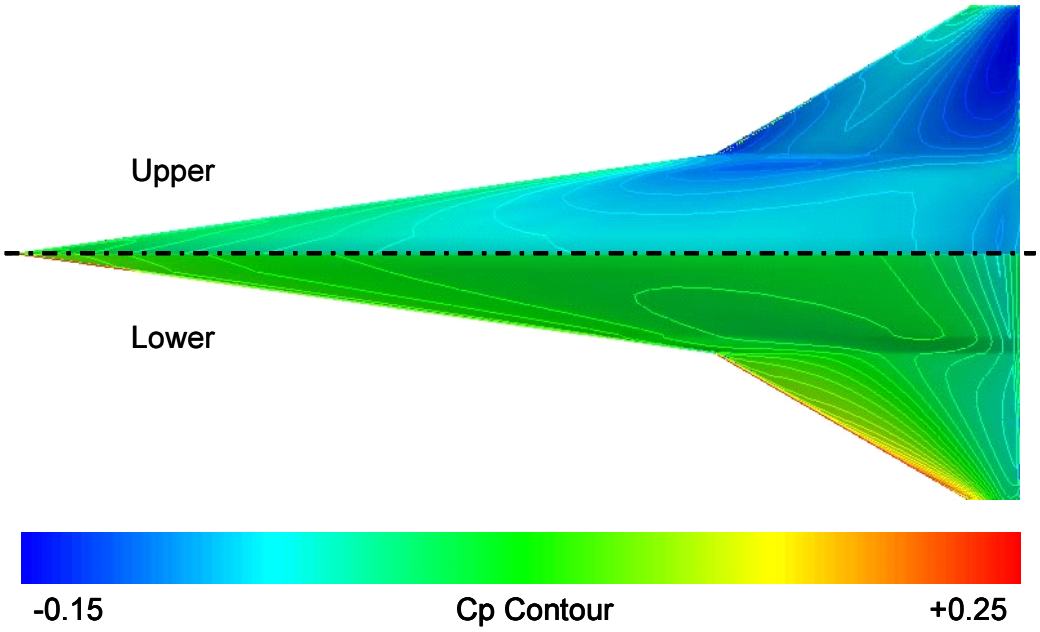


Figure 110: The C_p distributions around the initial and optimized wings (optimized through the SQP with the low-fidelity model and analyzed with the high-fidelity model).

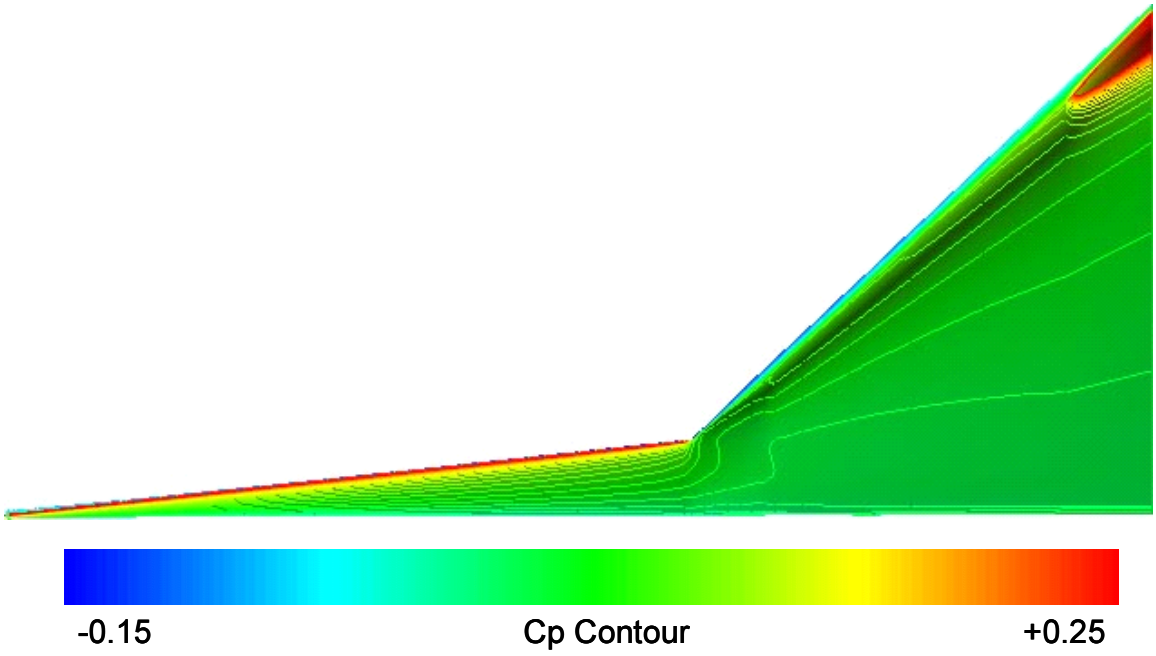


(a) Initial

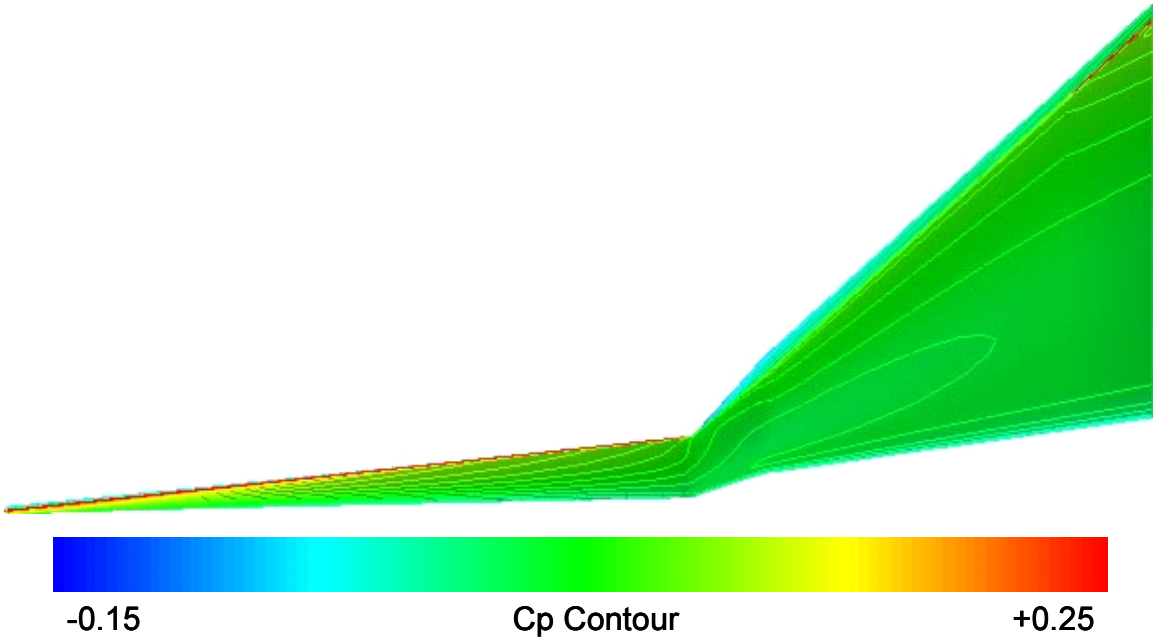


(b) Optimized

Figure 111: A top/bottom view of the C_p contours on the initial wing (analyzed with the high-fidelity model) and the optimized wing (optimized through the SQP with the low-fidelity model and then analyzed with the high-fidelity model).



(a) Initial



(b) Optimized

Figure 112: A front view of the C_p contours on the initial wing (analyzed with the high-fidelity model) and the optimized wing (optimized through the SQP with the low-fidelity model and then analyzed with the high-fidelity model).

5.3.4 Optimization with the High-Fidelity Model

The wing is optimized with the high-fidelity model by means of the SQP in MATLAB. Here, the same convergence criteria, set in the optimization through the SQP with the low-fidelity model, are used.

Figures 113, 114, and 115 show the histories of drag coefficients, lift coefficients, and volume, respectively. Here, volume is non-dimensional volume when the chord length at Section A is set to one. Initial and optimized aerodynamic coefficients, volume, and CPU time, and the number of function calls required to obtain an optimized wing are summarized in Table 33.

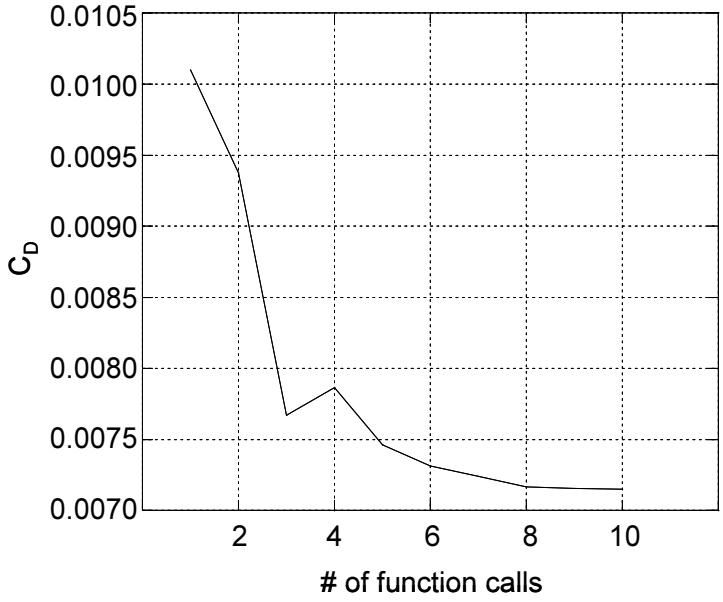


Figure 113: History of drag coefficients in the SQP with the high-fidelity model.

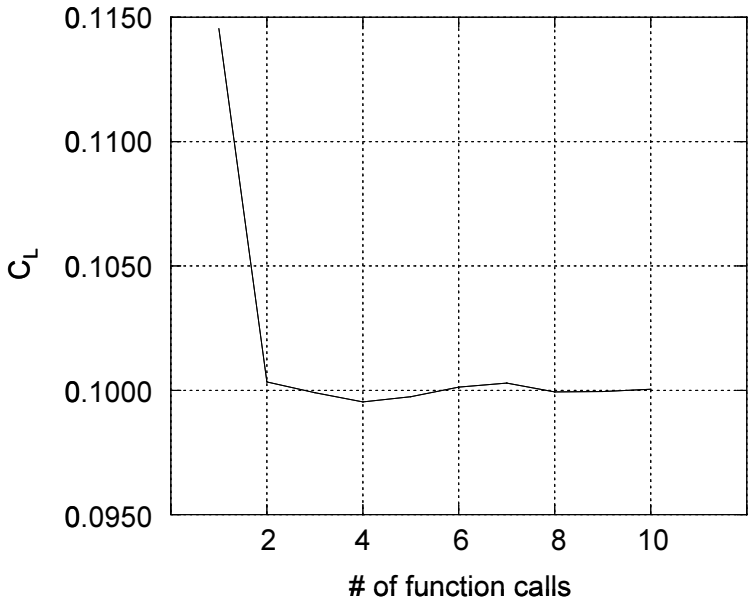


Figure 114: History of lift coefficients in the SQP with the high-fidelity model.

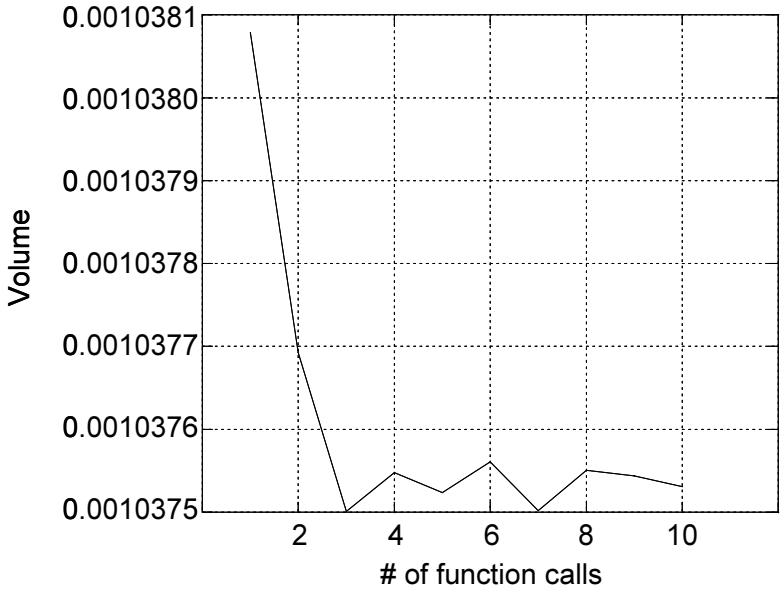


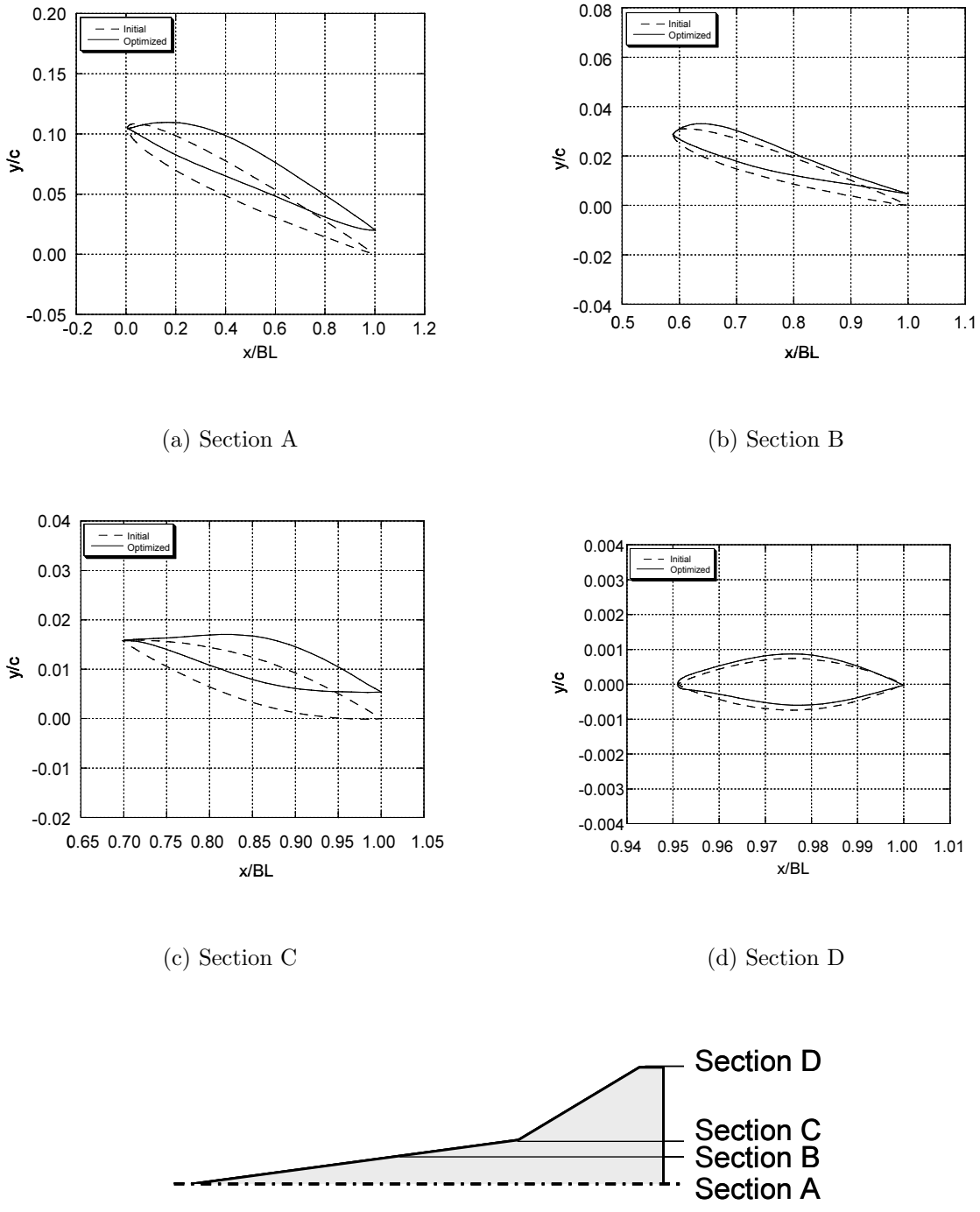
Figure 115: History of volume in the SQP with the high-fidelity model.

Table 33: Results with the high-fidelity model.

	C_D	C_L	Volume	No. of f_{high}, ∇f_{high} calls	CPU time [sec]
Initial	0.0107	0.1139	0.00104	1, 0	226
Optimized	0.0072	0.1000	0.00104	11, 10	47,810

Figure 116 shows the initial and optimized airfoils at sections A, B, C, and D. Since the scales of the horizontal and vertical axes in Figures 116(a) differ, 116(b), 116(c), and 116(d), one should note that the angles of attack shown in these figures differ from the true ones. Figure 117 shows the C_p distributions around the initial and optimized airfoils at sections A, B, C, and D. Figures 118 and 119 show a top/bottom view and a front view of the C_p contours on the surfaces of the initial and optimized wings, respectively, visualized with FieldView [1].

Regarding the drag mitigation mechanism, similar phenomena exhibited in the low-fidelity case have also been observed in the high-fidelity case. However, a comparison of Figures 111 and 118 shows that compression on the leading edge of the lower surface from Sections C to D shrinks more with the high-fidelity case than with the low-fidelity case.



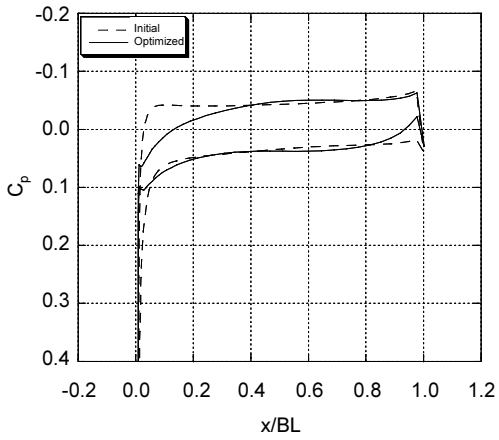
(a) Section A

(b) Section B

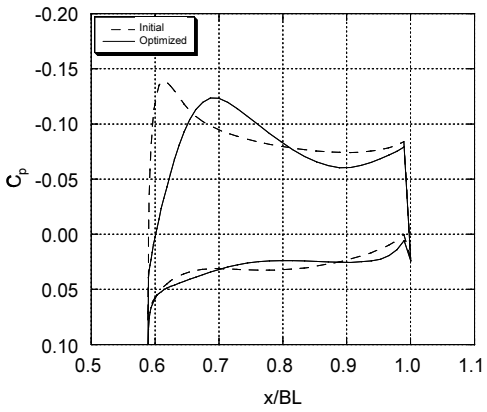
(c) Section C

(d) Section D

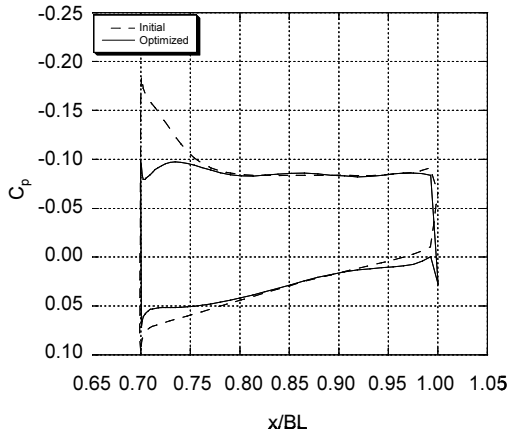
Figure 116: Shapes of the initial and optimized wing sections (optimized through the SQP with the high-fidelity model).



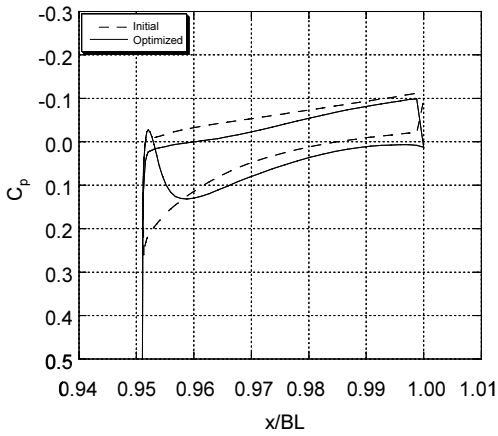
(a) Section A



(b) Section B



(c) Section C



(d) Section D

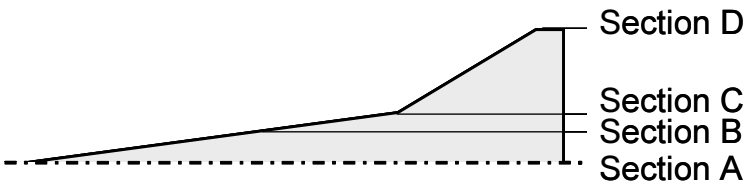
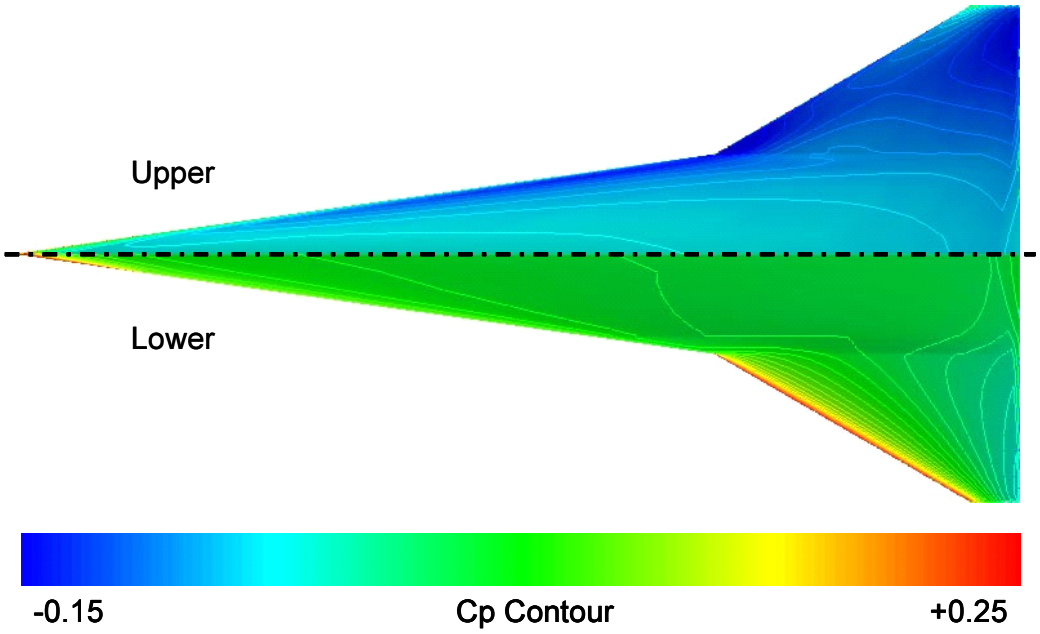
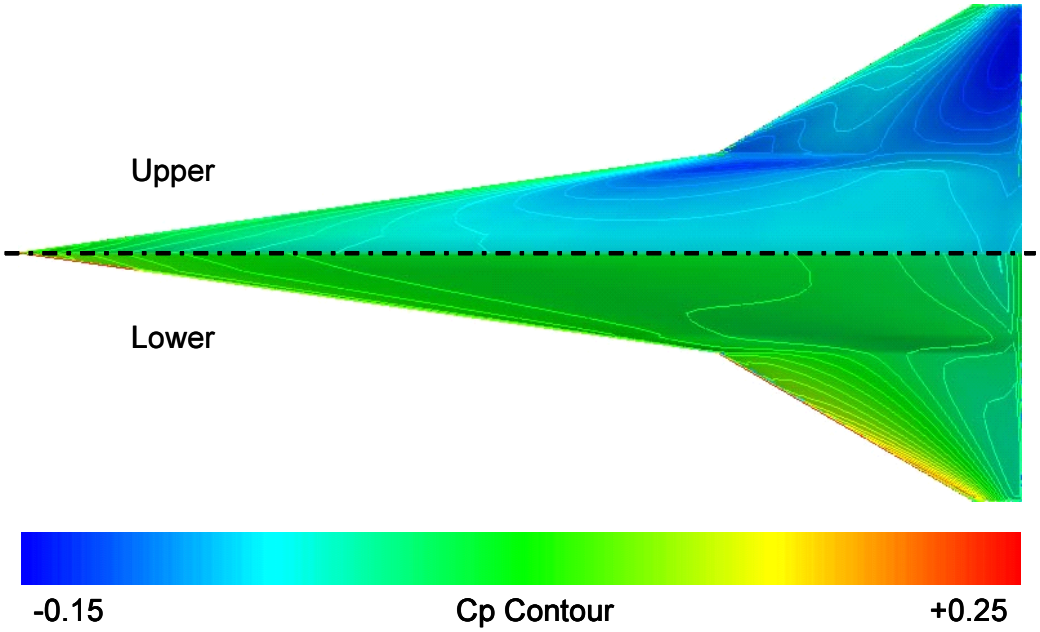


Figure 117: The C_p distributions around the initial and optimized wings (optimized through the SQP with the high-fidelity model).

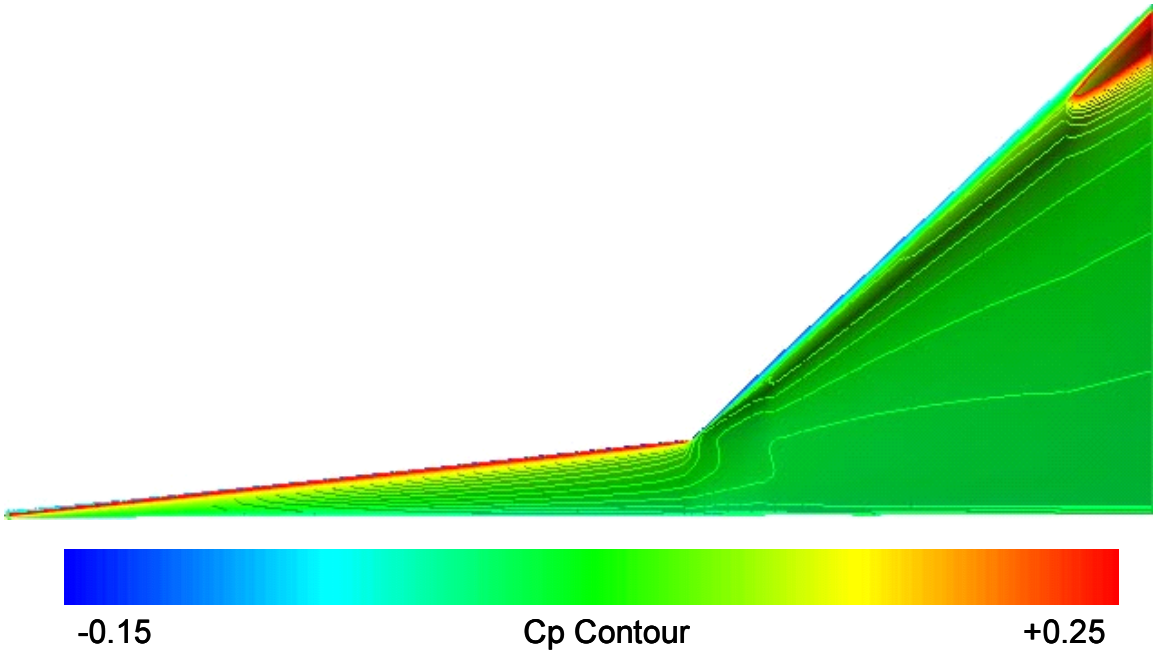


(a) Initial

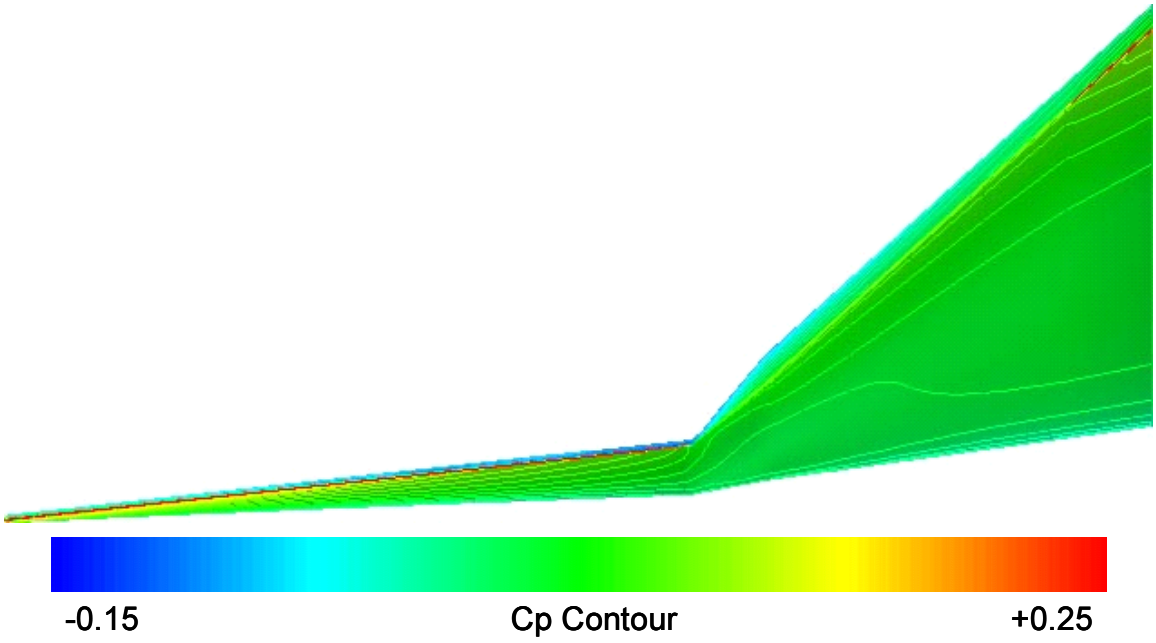


(b) Optimized

Figure 118: A top/bottom view of the C_p contours on the initial (analyzed with the high-fidelity model) and optimized wings (optimized through the SQP with the high-fidelity model).



(a) Initial



(b) Optimized

Figure 119: A front view of the C_p contours on the initial wing (analyzed with the high-fidelity model) and the optimized wing (optimized through the SQP with the high-fidelity model).

5.3.5 Optimization with the Robust AMF

The wing is optimized with the Robust AMF. Here, the termination tolerance of the function value is set to 10^{-6} (Equation 29), that of the design variables 10^{-6} (Equation 30), and that of the constraint violation in the governing equation of the trust region ration 10^{-3}.

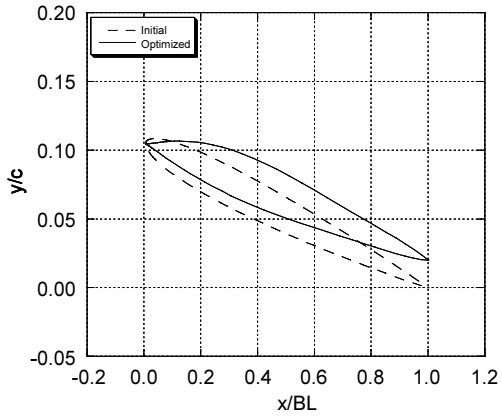
Table 34 shows a convergence history with the Robust AMF in which the numbers of function calls are cumulative. As shown in Table 34, the C_L, which is a scaled constraint in the inner optimization in the AMF (Figure 9) and the only constraint involved in Equation 57, satisfies the required condition during the entire optimization process. Therefore, optimization with the Robust AMF, which is more robust than the original AMF when constraints are violated, should be the same as that with the original AMF.

Table 34: Convergence history with the Robust AMF.

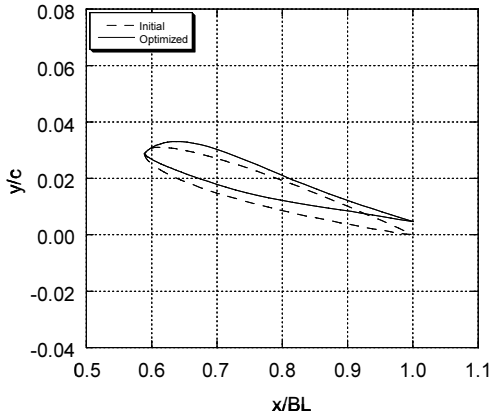
Iter	f_{high}	g_{high_1}	g_{high_2}	No. of function calls			
	C_D	C_L	Volume	f_{high},	f_{low},	∇f_{high}	∇f_{low}
0	0.0101	-0.0145	0.0000	1,	1,	0,	0
1	0.0077	-0.0011	0.0000	2,	5,	1,	4
2	0.0076	-0.0004	0.0000	3,	9,	2,	7
3	0.0073	-0.0002	0.0000	4,	13,	3,	10
4	0.0073	0.0000	0.0000	5,	17,	4,	13

Figure 121 shows the initial and optimized airfoils at sections A, B, C, and D. Since the scales of the horizontal and vertical axes in Figures 120(a), 120(b), 120(c), and 120(d) are different, one should note that the angles of attack shown in these figures are different from the true ones. Figure 121 shows the C_p distributions around the initial and optimized airfoils at sections A, B, C, and D. Figures 122 and 123 show a top/bottom view and a front view of the C_p contours on the surfaces of the initial and optimized wings, respectively, visualized with FieldView [1]. Similar phenomena seen with the different optimization methods have been observed with the AMF case

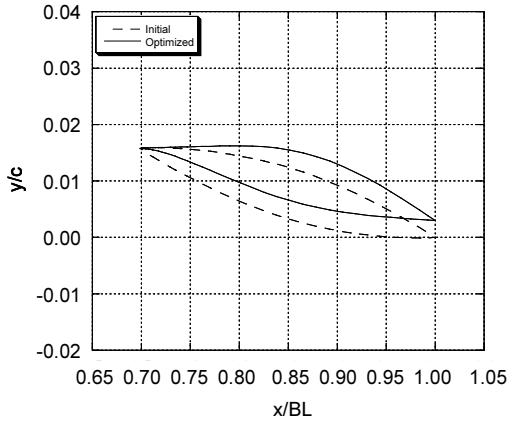
as well.



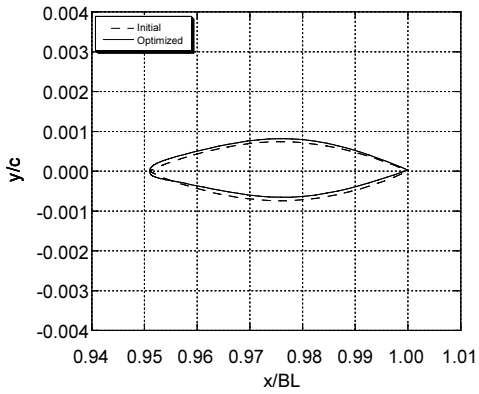
(a) Section A



(b) Section B



(c) Section C



(d) Section D

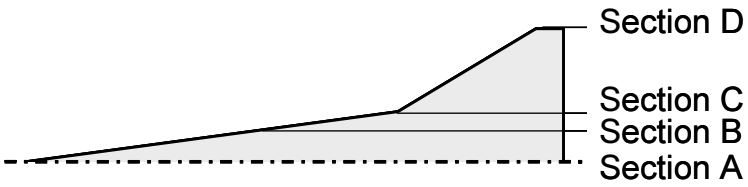
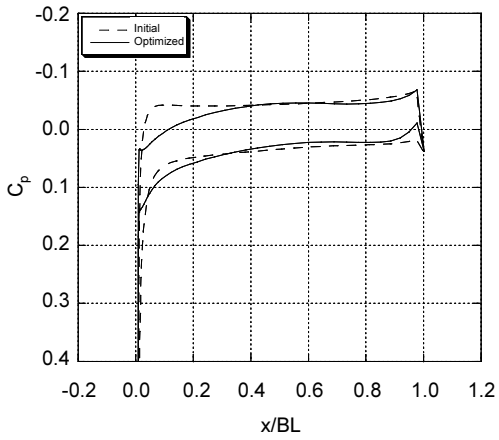
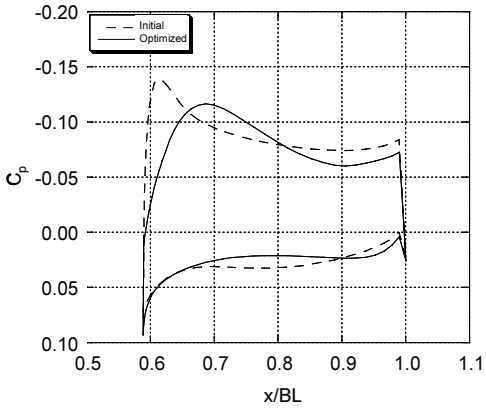


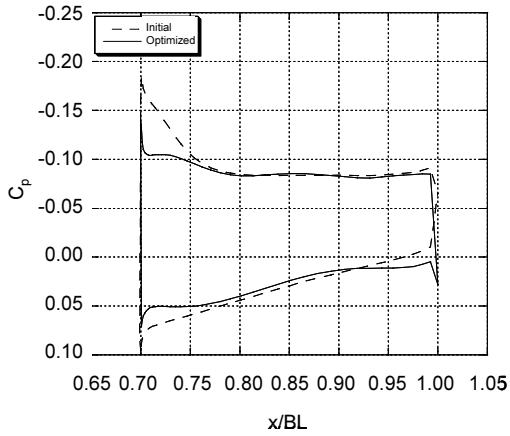
Figure 120: Shapes of the initial and optimized wing sections (optimized through the Robust AMF).



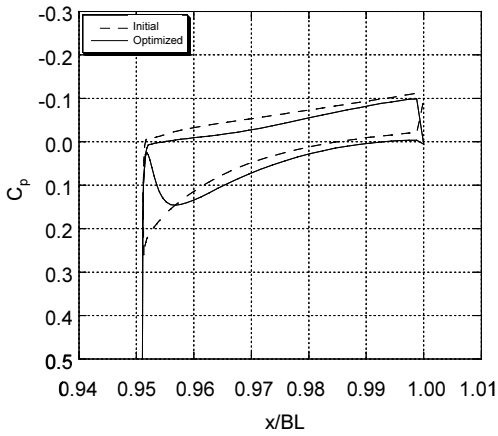
(a) Section A



(b) Section B



(c) Section C



(d) Section D

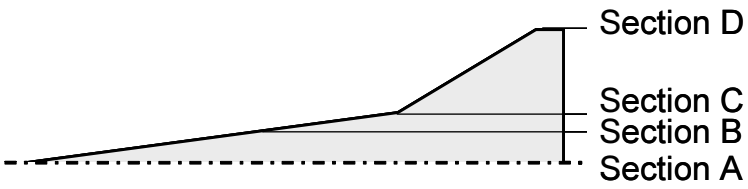
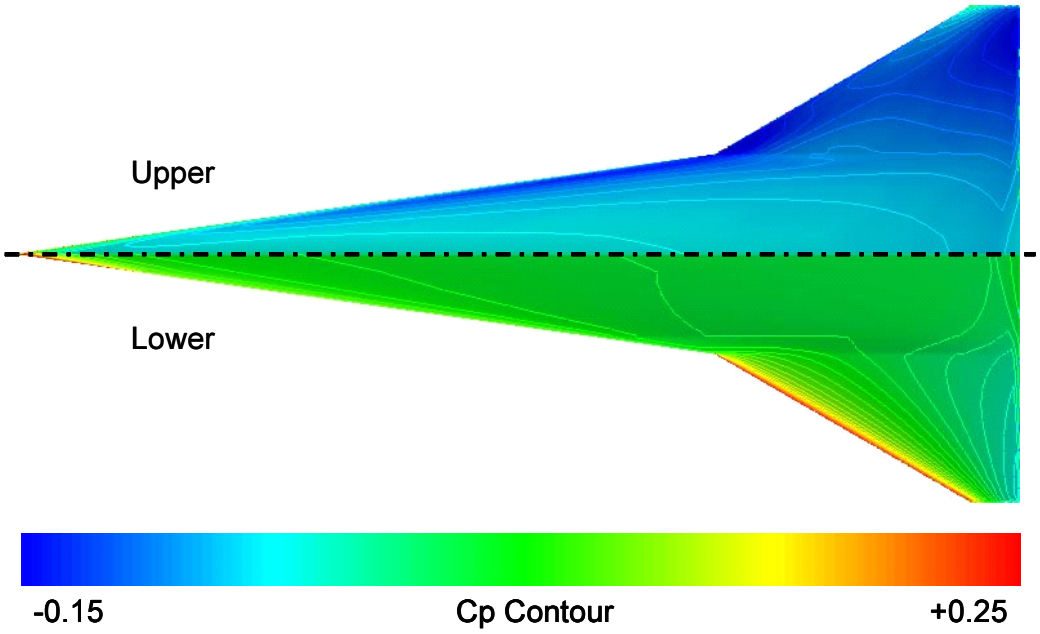
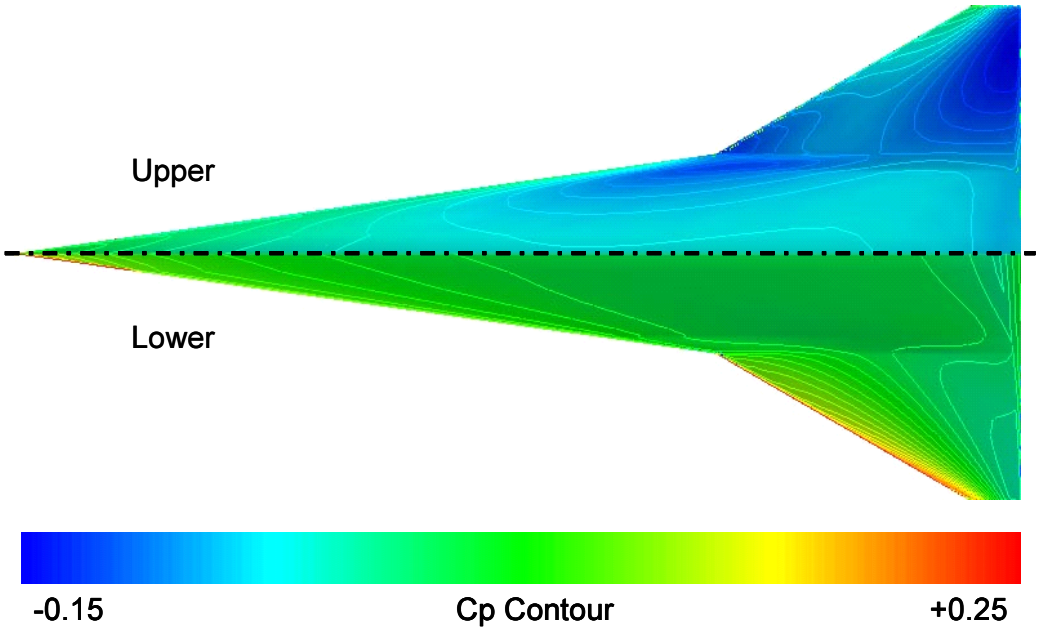


Figure 121: The C_p distributions around the initial and optimized wings (optimized through the Robust AMF).

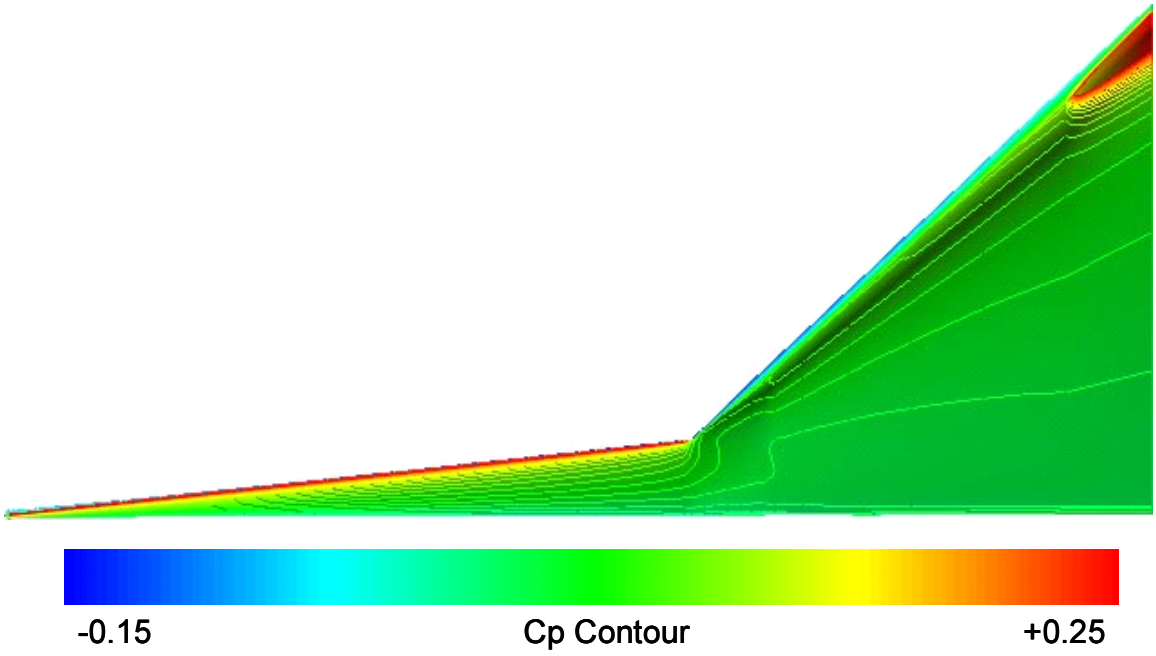


(a) Initial

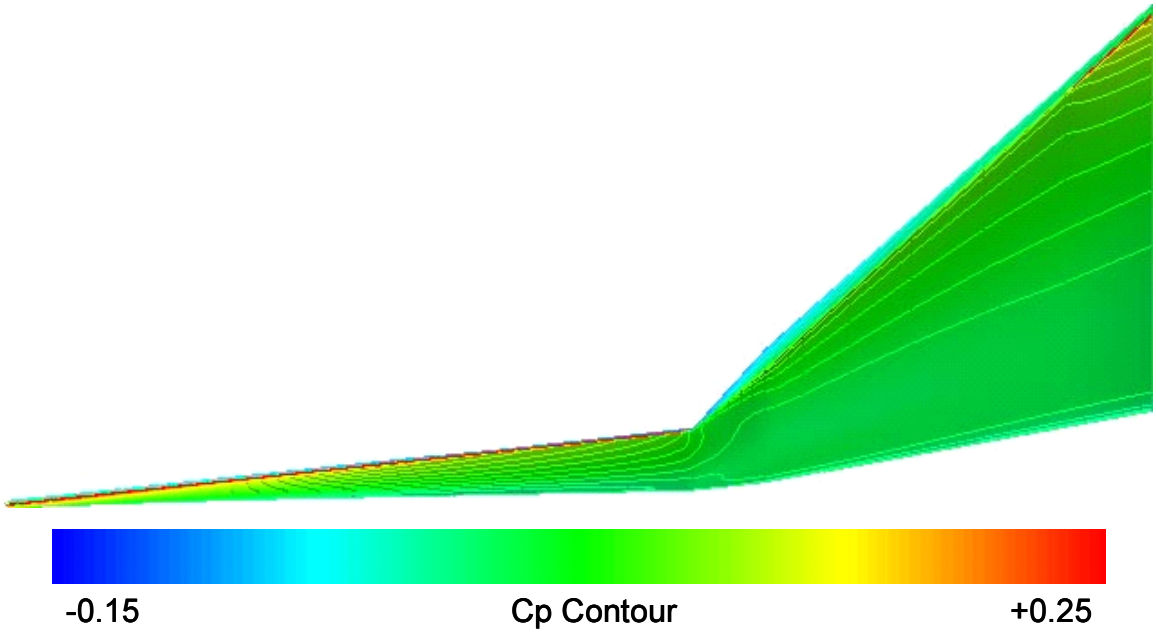


(b) Optimized

Figure 122: A top/bottom view of the C_p contours on the initial wing (analyzed with the high-fidelity model) and the optimized wing (optimized with the Robust AMF).



(a) Initial



(b) Optimized

Figure 123: A front view of the C_p contours on the initial wing (analyzed with the high-fidelity model) and the optimized wing (optimized with the Robust AMF).

5.3.6 Optimization with the Original AMF

As the previous section mentioned, optimization with the Robust AMF and that with the original AMF were the same in this problem because none of the constraints violated the given requirements during the entire optimization process.

5.3.7 Summary of the Design of a Wing in the Transonic Speed Regime

All the optimization results conducted with the various methods for the design of a wing in the supersonic speed regime are summarized in Table 35. In columns "C_L" and "Volume," the values in parentheses represent the degree of violation against the constraints, and in column "CPU time," the values in parentheses represent non-dimensional CPU time normalized by the CPU time required for optimization through the SQP with the high-fidelity model.

Table 35: Comparison of convergence history with different optimization methods.

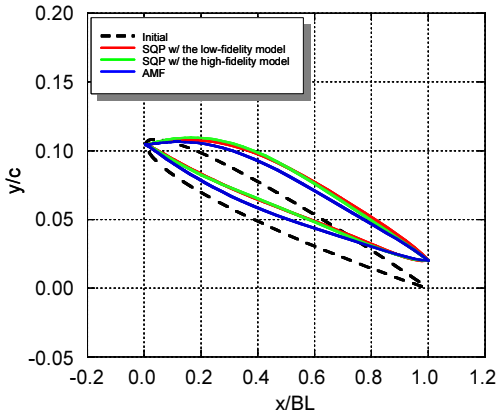
	C_D	C_L	Volume	No. of f_{high},	f_{low},	∇f_{high},	∇f_{low} calls	CPU time [sec]
SQP with f_{high}	0.0072	0.1000	0.00104	11,	0,	10,	0	47,810
		(0.0000)	(0.00000)					(1.00)
SQP with f_{low}	0.0075	0.1024	0.00104	1,	7,	0,	7	1,891
		(-0.0024)	(0.00000)					(0.04)
Robust AMF	0.0073	0.1000	0.00104	5,	17,	4,	3	17,097
		(0.0000)	(0.00000)					(0.36)
Original AMF	0.0073	0.1000	0.00104	5,	17,	4,	13	17,097
		(0.0000)	(0.00000)					(0.36)
Initial	0.0101	0.1144	0.00104	1,	0,	0,	0	226

From Table 35, following remarks can be deduced:

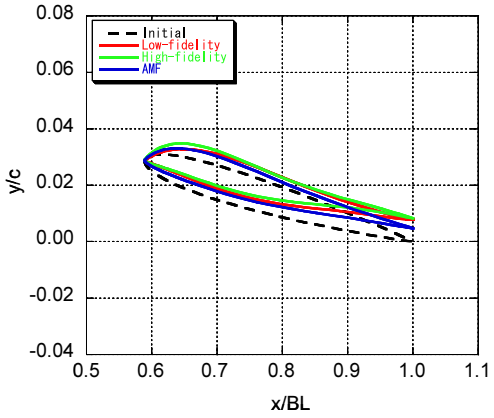
- The SQP with the high-fidelity model works best in terms of the optimized value, but it requires the most CPU time.

- Robust AMF is the same as the original AMF in this particular problem because the scaled constraints are never violated during the entire optimization process.
- Robust AMF and the original AMF are competitive against optimization through the SQP with the high-fidelity model because the required CPU time of the former is 36% of that of the latter, and the optimized value is slightly higher than that of the latter.
- The SQP with the low-fidelity model works well in this study. However, this finding is coincidental because an optimized wing with the low-fidelity model does not guarantee that it also satisfies constraints with the high-fidelity model.

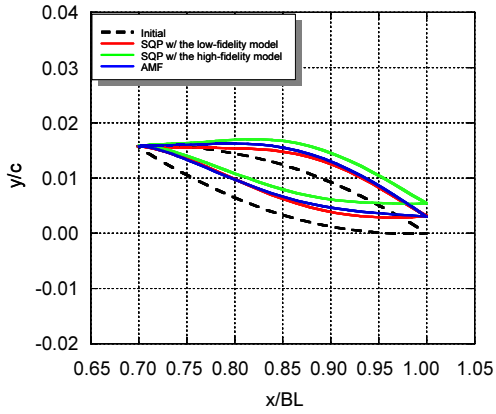
Finally, wing sections optimized with different methods and their distributions of pressure coefficients are compared in Figures 124 and 125. The shapes of the optimized wing sections have some diversity, as seen in Figures 124, and the distributions of the pressure coefficients are almost identical, as seen in Figure 125. In Figure 124(b), the AMF yields a slightly different wing section from the other methods while it yields similar C_p distributions in Figure 125(b). This may be caused by the three-dimensional effect of the flow field around the wing. All optimized wings reduce the wave drag with the same mechanism mentioned previously.



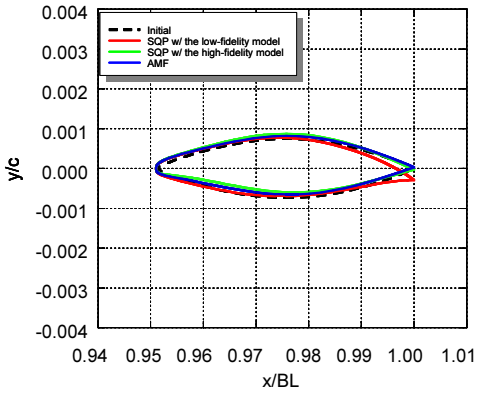
(a) Section A



(b) Section B



(c) Section C



(d) Section D

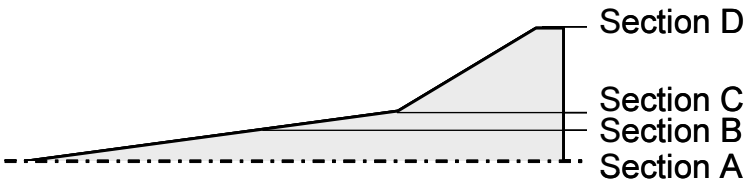
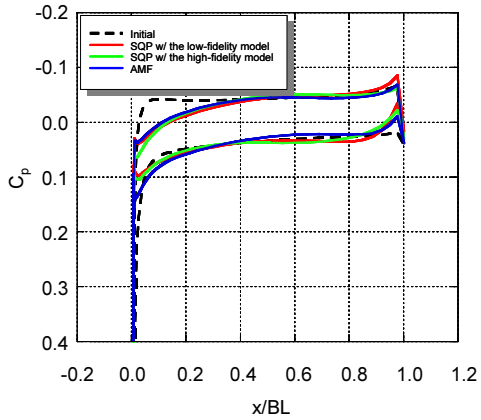
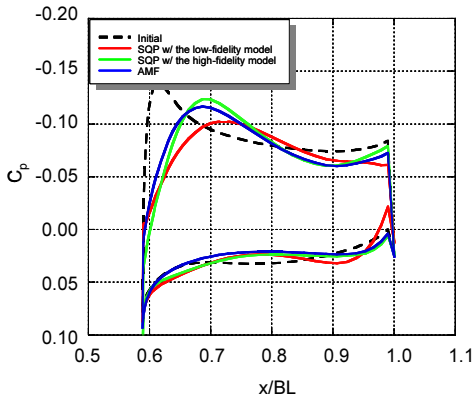


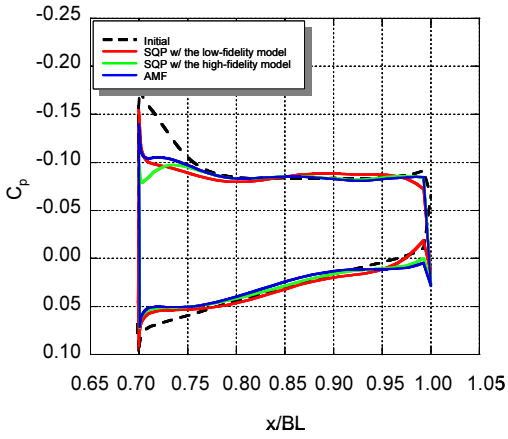
Figure 124: Shapes of the initial and optimized wing sections.



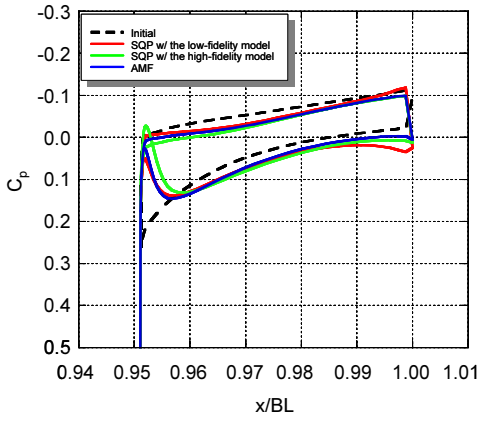
(a) Section A



(b) Section B



(c) Section C



(d) Section D

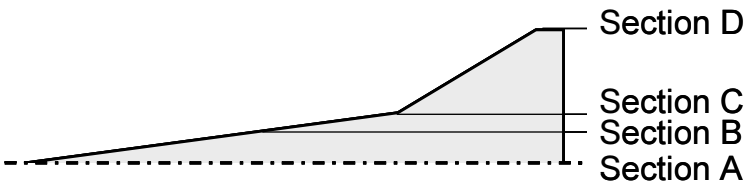


Figure 125: The C_p distributions around the initial and optimized wings.

5.4 Supplemental Study in the Design of a Wing in the Supersonic Speed Regime

In the previous section, the wing was optimized with several optimization methods. However, since the low- and high-fidelity models compute almost the identical aerodynamics shown in Figures 94 and 94, the low-fidelity model is modified so that it has fewer grid points and thus computes different aerodynamics from the high-fidelity model.

Figure 126 shows the grid of the new low-fidelity model, which has 8,649 nodes (31 in the chord-wise direction, 31 in the circumferential direction, and 9 from the body to the outer boundary). Figures 127 and 128 show comparisons of the high-fidelity model with the new low-fidelity model in C_L and C_D at Mach 1.98. Although PANAIR is not used in the present optimization problem, the results of the analysis results are also shown in Figures 127 and 128. As seen clearly especially, in Figure 127, the low- and high-fidelity models compute different aerodynamics.

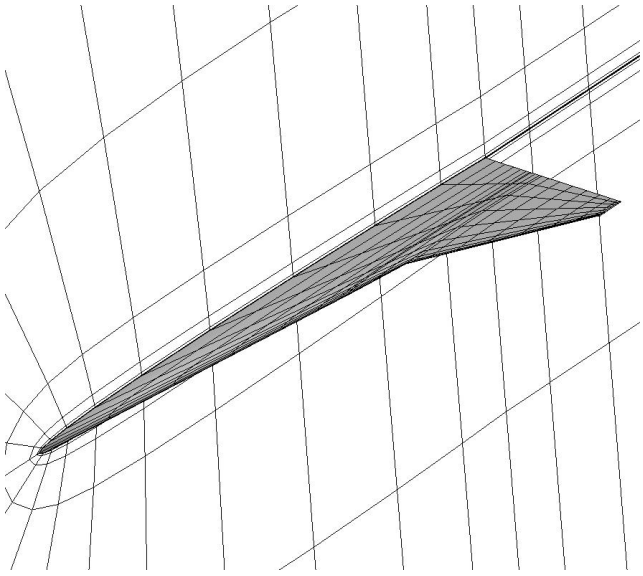


Figure 126: New Coarse grid around the wing ($31 \times 31 \times 9$) in the new low-fidelity model.

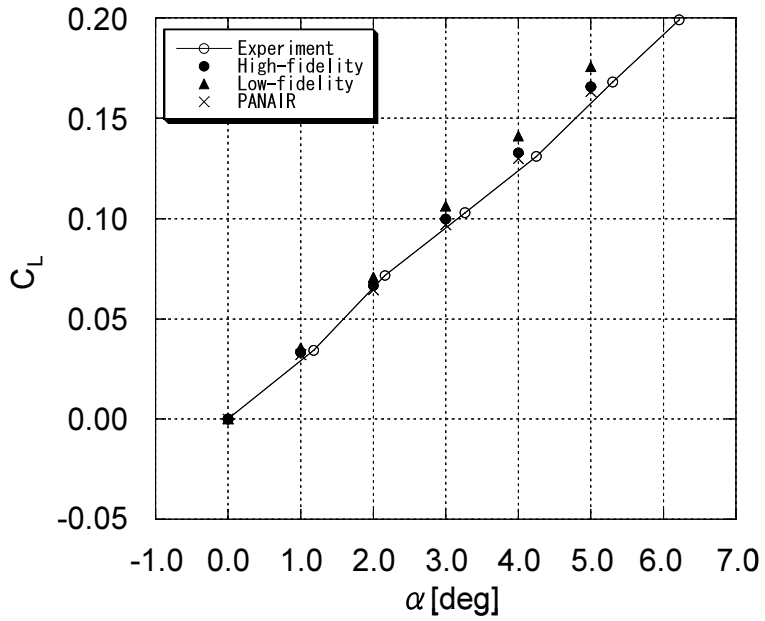


Figure 127: Comparisons of the lift coefficient.

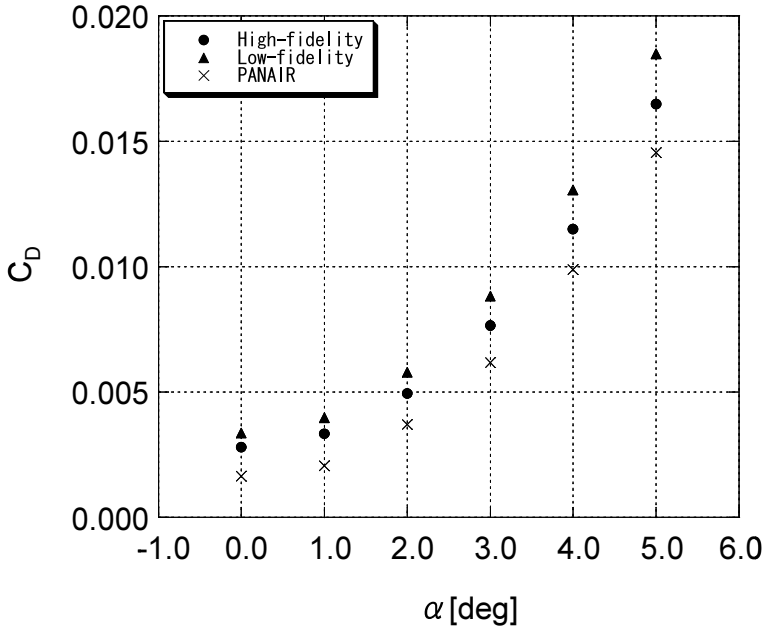


Figure 128: Comparisons of the drag coefficient.

The optimization problem is almost identical to that of the previous case, but the

author enforces a greater C_L than the initial C_L. Therefore, the optimization problem of a wing at Mach 2.0 with an angle of attack of 0.0 [deg] can be defined as follows this time:

$$\begin{aligned}
\underset{\beta}{\text{Minimize}} \quad &: \quad C_D, && (73)\\
\text{Subject to} \quad &: \quad C_L \geq C_{L,initial}, && (74)\\
&\quad\quad Wing\,volume \geq Wing\,volume_{initial}, && (75)\\
&\quad\quad 0.03 \leq Airfoil\,thickness \leq 0.04, && (76)\\
&\quad\quad -2.0 \leq Twist_D \leq Twist_C \leq Twist_B \leq Twist_A \leq 7.0, && (77)\\
&\quad\quad Twist_B - Twist_C \leq 1.0. && (78)
\end{aligned}$$

In the AMF cases, the termination tolerance of the function value is set to 10^{-6} (Equation 29), that of the design variables 10^{-6} (Equation 30), and that of the constraint violation in the governing equation of the trust region ration 10^{-3}. In the SQP case, the termination tolerance of the function value is set to 10^{-6}, that of the design variables 10^{-6}, and that of the constraint violation 10^{-3}.

All the optimization results conducted with the various models for the design of a wing in the supersonic speed regime are summarized in Table 36. In columns "C_L" and "Volume," the values in parentheses represent the degree of violation against the constraints, and in column "CPU time," the values in parentheses represent non-dimensional CPU time normalized by the CPU time required for optimization through the SQP with the high-fidelity model.

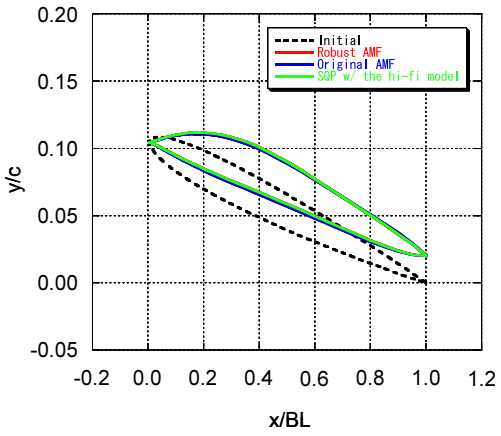
Table 36: Comparison of results with different optimization methods.

	C_D	C_L	Volume	No. of f_{high},	f_{low},	∇f_{high},	∇f_{low} calls	CPU time [sec]
SQP with f_{high}	0.0087	0.1145	0.00104	10,	0,	9,	0	31,003
		(0.0000)	(0.00000)					(1.00)
Robust AMF	0.0089	0.1142	0.00104	6,	67,	3,	64	19,407
		(+0.0003)	(0.00000)					(0.63)
Original AMF	0.0088	0.1145	0.00104	13,	47,	5,	35	25,042
		(0.0000)	(0.00000)					(0.81)
Initial	0.0100	0.1145	0.00104	1,	0,	0,	0	226

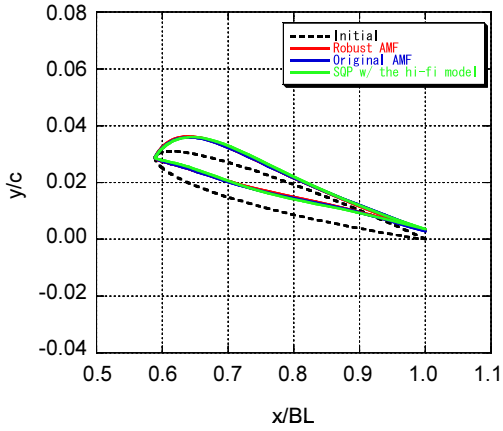
From Table 36, the following remarks can be deduced:

- The SQP with the high-fidelity model works the best in terms of the optimized value, but it takes the longest in CPU time.
- The Robust AMF terminates the optimization more rapidly than any other case. However, the final C_L only slightly violates the given constraint.
- The original AMF used in this study requires the largest number of high-fidelity function calls. However, since the number of high-fidelity derivative function calls is less than that of the SQP with the high-fidelity model, the total CPU time that the original AMF requires is still less than that of the SQP with the high-fidelity model.

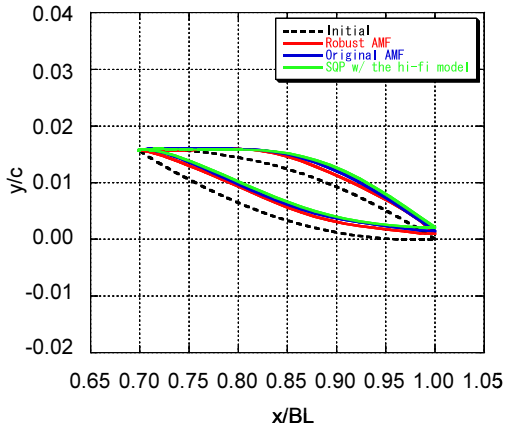
Finally, wing sections optimized with different methods and their distributions of pressure coefficients are compared in Figures 124 and 125. From these figures, one can conclude that all optimization methods find similar design points as optimized design points. In addition, the mechanism of how to reduce drag is the same as that of the previous wing design case.



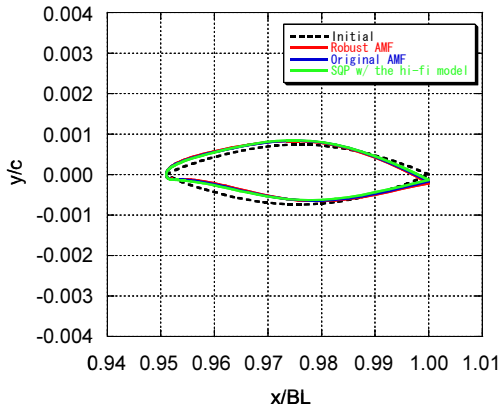
(a) Section A



(b) Section B



(c) Section C



(d) Section D

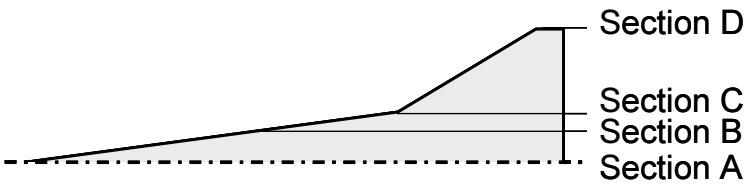
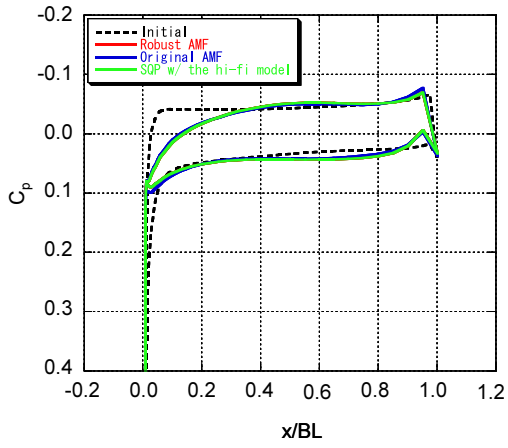
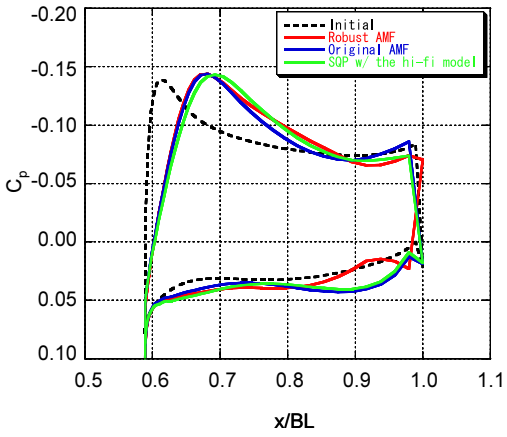


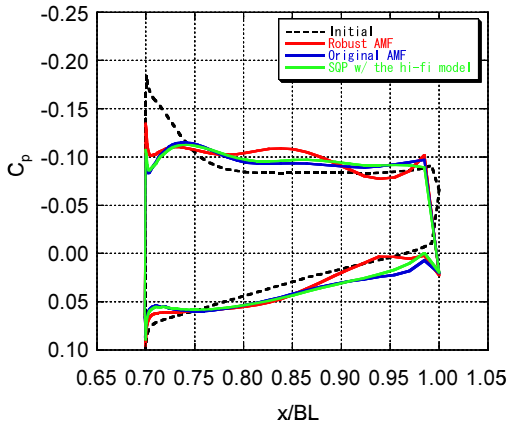
Figure 129: Shapes of the initial and optimized wing sections.



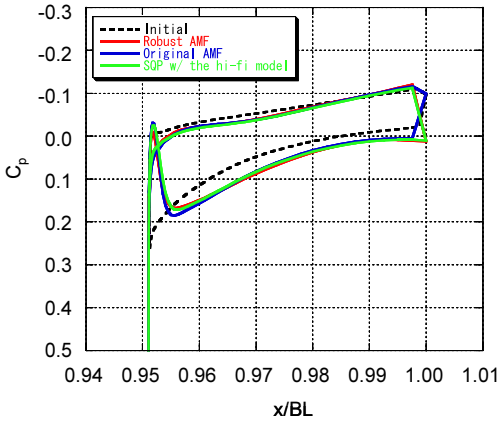
(a) Section A



(b) Section B



(c) Section C



(d) Section D

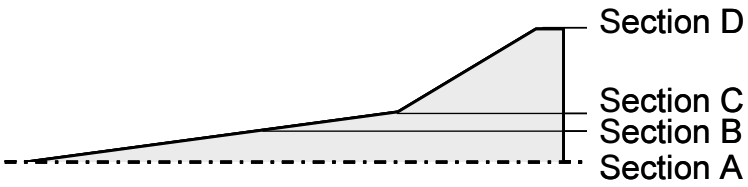


Figure 130: The C_p distributions around the initial and optimized wings.

Since the constraint is violated in the Robust AMF case, the methods proposed in Subsections 5.2.2, 5.2.3 are implemented.

The wing optimized with the Robust AMF is manually rotated so that it produces a C_L value equivalent to that of the initial wing. In this case, the angle of attack is 0.01 [deg], and the corresponding C_D and C_L values are 0.0090 and 0.1145, respectively.

The constraint is tightened by the amount of $tolg(= 0.001)$. The result is summarized in Table 37, in which the result obtained under the original constraint is also presented.

Table 37: Comparison of results with different constraints in the Robust AMF.

	C_D	C_L	Volume	No. of f_{high}, f_{low}, ∇f_{high}, ∇f_{low} calls	CPU time [sec]
$C_L > C_{L,initial}$	0.0089	0.1142 (+0.0003)	0.00104 (0.00000)	6, 67, 3, 64	19,407
$C_L > C_{L,initial} + tolg$	0.0090	0.1152 (-0.0007)	0.00104 (0.00000)	6, 56, 3, 53	17,708
Initial	0.0100	0.1145	0.00104	1, 0, 0, 0	226

The wing optimized with the Robust AMF with the tightened constraint is manually rotated so that it yields a C_L value equivalent to that of the initial wing. In this case, the angle of attack is -0.027 [deg], and the corresponding C_D and C_L values are 0.0089 and 0.1145, respectively.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

Chapter I identified the necessity of the VFO-type approach in modern aerospace engineering, and Chapter II reviewed related literature. Based on the findings of the literature, Chapter III focused on the AMF optimization method and revealed several problems in its applications to some simple analytical problems. Section 3.3 presented research questions emerging from these problems, and Chapter IV mentioned hypotheses that would address the research questions.

In closing, a discussion about the contributions of this dissertation with regard to each of the research questions is presented and followed by a discussion about future work.

6.1 Answering the Research Questions

The research questions and the hypotheses are restated below, and the answers to each research question are given based on the accomplishments of this dissertation.

6.1.1 Research Question 1

Question 1: *How can one obtain correct derivatives, even in noisy functions, in a shorter time?*

Generally, high-fidelity models such as the CFD contain numerical noise, and their derivatives, computed with the FD method, are often deteriorated. In addition, the computational time required for calculating derivatives is proportional to the

number of design variables with the FD method, so one can not deal with many design variables either.

Hypothesis 1: *Use the automatic differentiation (AD) technique to compute derivatives.*

The AD technique, which reads codes of analysis models and automatically generates new derivative codes based on the rules mentioned in 4.1.3.1 and 4.1.3.2, is applied. If derivatives are computed with the generated derivative code by the AD, they are analytical, and the required computational time is almost independent of the number of design variables, which is advantageous to realistic aerospace engineering problems. However, if analysis models implement iterative computations such as the CFD, which solves system partial differential equations iteratively, computing derivatives directly through the derivative code generated with the AD tool requires massive memory size. The author solved this deficiency by modifying the derivative code systematically, explained in 4.2, and successfully applied the AD to general CFD software.

6.1.2 Research Question 2

Question 2: *The governing equation of the trust region ratio does not work properly when an optimization problem has constraints, leading to inaccurate behavior in the original AMF. How can one modify the governing equation of the trust region ratio even if the optimization problem has constraints?*

The original form of the governing equation of the trust region ratio is very strict

against the violation of constraints; thus, as shown in 3.2.2, it often computes unrealistic values even though surrogate functions generated are tolerable. These unrealistic values in the trust region ratio cause the size of the trust region to shrink, eventually leading to the erroneous termination of optimization with the AMF.

Hypothesis 2: *Modify the governing equation of the trust region ratio so that it can accept violations of constraints within some tolerance.*

The governing equation of the trust region ratio is modified, shown in 4.3, where only one term deals with violations of constraints within some tolerance, and the remaining terms are independent of the violations of constraints. With these modifications, the AMF can continue optimization without terminating it erroneously and eventually find the optimum design point. In addition, if constraints are not violated during the entire optimization process, the modified governing equation of the trust region ratio works exactly the same as the original, shown in the demonstration conducted in Section 5.3.

6.2 Summary of Contributions

The primary objective of this research has been the development of a new VFO framework that is more robust than the existing ones. The major contributions of this dissertation are that it clarifies the method of how to apply the AD to the CFD analysis and it generates a Robust AMF in which the derivatives are computed through the AD and in which the trust region management is controlled by the new governing equation of the trust region ratio.

- Clarifying the method of how to apply the AD for CFD analysis

  As long as a code of the CFD solver is written in a computer language that is

compatible with an available AD tool (see Table 6), one can create a derivative code very rapidly by means of the AD tool and the method proposed in this dissertation. Since the computational time for calculating the derivatives through this derivative code is analytical and independent of the number of design variables, one can be free from choosing the proper size of the step size, which is required when using the FD, and explore the multi-dimensional design space in a shorter time. Moreover, since CFD is based on the system PDEs, one can obtain derivatives of any engineering analyses based on the system PDEs by following the same method clarified in this dissertation. This is very prominent when gradient-based optimization is conducted.

- Robust AMF

  The Robust AMF utilizes the new governing equation of the trust region ratio, which can accept a violation of the constraint depending on the amount of $tolg$ during the entire optimization process. Thus, the CPU time required for the optimization may be less than the CPU time required for not only the SQP but also the original AMF, and the final design can be closer to that of the SQP than that of the original AMF. Indeed, for the airfoil design in the transonic speed regime in Section 5.1, the Robust AMF requires 60% of the CPU time required of the SQP while the original AMF requires 86% of the CPU time, and the airfoil shape optimized by the Robust AMF is more similar to that by the SQP than that by the original AMF. On the other hand, for the wing design in the supersonic speed regime in Section 5.3, the Robust AMF and the original AMF work exactly the same, requiring less than half of CPU time than the SQP, and their final designs are very similar to that obtained by the SQP. From these results, if a slight violation within the amount of $tolg$ is acceptable in the final design, the author recommends the use of the Robust AMF as it can work exactly the same or better than the original AMF. However, since the

Robust AMF does not guarantee that the final design point will be located inside the feasible region, it has a significant weakness for optimization problems, in which the final design point must be inside the feasible region. Fortunately, in the airfoil and wing design conducted in this dissertation, this problem can be solved by some methods proposed in Section 5.2. Especially, the method of tightening the constraint by the amount of *tolg* proposed in Subsection 5.2.3 may be promising because it not only guarantees that the final design point will be located inside the feasible region but it also allows wider exploration during the optimization than the original AMF.

6.3 Future Work

Through the implementations of the Robust AMF, several things that desired to be modified in future work have been revealed. They are discussed in this section.

6.3.1 Tuning Derivative Codes Generated by the AD Method

This study has clarified the advantages of the AD for CFD: The AD tool generates a derivative code without significant computational or mathematical pre-processing. Now, comparing the AD and adjoint method mentioned in 4.1.2 is very interesting because the amount of the computational and mathematical pre-processing is so different between them although they should compute the same derivative values if they are used in the same problem.

Using the adjoint method, Kim [50] optimized the shape of a wing and reported required memory size and time per iteration in his study, both of which are compared with those obtained using the AD method in this study. Table 38 presents the values normalized by those measured with flow solvers for the purpose of fair comparisons (i.e., if a flow solver requires 100MB and an adjoint code requires 300MB, the normalized memory size required by the adjoint code is 3.0, and if a flow solver requires

3.0 seconds per iteration and an adjoint code requires 15.0 seconds per iteration, the normalized time per iteration required by the adjoint code is 5.0).

Table 38: Comparison of the AD and adjoint methods in required memory size and time per iteration.

	Memory size	Time per iteration
Adjoint [50]	2.25	7.07
AD (airfoil optimization with the low-fidelity model)	6.18	9.24
AD (airfoil optimization with the high-fidelity model)	5.77	7.60
AD (wing optimization in with the low-fidelity model)	4.64	4.63
AD (wing optimization in with the high-fidelity model)	5.23	7.09

The above table also shows that the derivative codes generated by the AD method require much more memory size than those generated by the adjoint method. This finding indicates that the derivative codes generated by the AD method waste memory on unnecessary values during derivative computations. Therefore, if one can identify the unnecessary parts of the derivative codes in future work, the computational time for derivative calculation can be shortened. If such a modification proves too difficult, the use of a computer with large memory is recommended.

In addition, Kim [50] pointed out that the longer computational time per iteration in the adjoint code results from poor vectorization of the adjoint code. Here, vectorization represents the process of converting a computer code from scalar to vectorized implementation (i.e., a single instruction can perform multiple operations, depicted in Figure 131). Vector processing is a major feature of both conventional and modern supercomputers. Currently, the derivative codes generated by the AD method, shown in Table 38, consume four to ten times as much computational time as the flow solvers in this study. Since this study has not focused on applying vectorization for the derivative codes, speeding up computational time through vectorization is feasible

for the derivative codes generated by the AD method as well. Thus, to implement vectorization, one should modify the structure of computer codes manually or use automatic vectorization. Like automatic differentiation, this approach has been the main focus of considerable research in computer science.

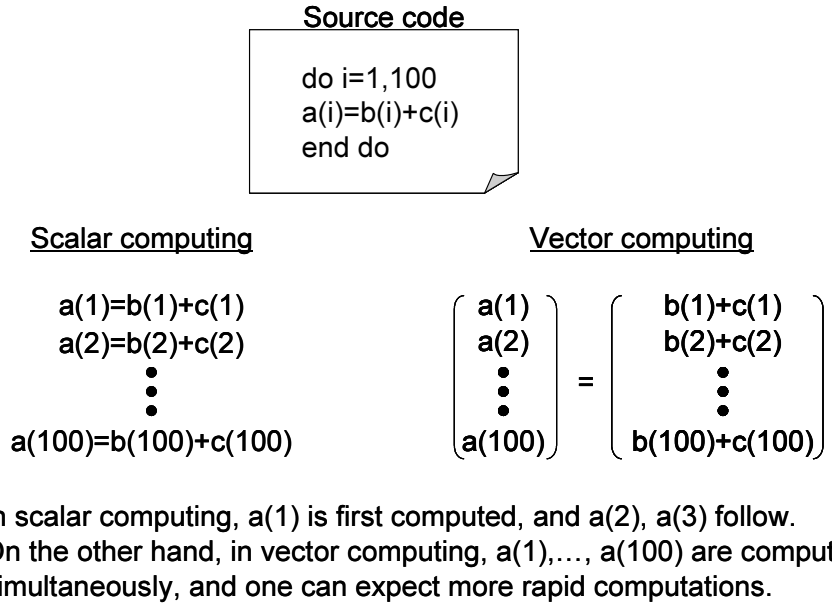


Figure 131: The difference between scalar and vector computing.

6.3.2 Obtaining the Optimum Design Point in a Feasible Region with the AMF

The modified governing equation of the trust region ratio permits user-specified violations of constraints, mentioned in Section 4.3. With this modification, the Robust AMF, unlike the original AMF, does not lead to a converged design point stuck in an erroneous location. The author expected this modification to prevent the erroneous termination of optimizations with the AMF and the optimized design point to lie inside the feasible region because surrogate functions created near constraints (i.e., $g = 0$) could mimic the true functions around the constraints (i.e., $g = 0$), as depicted in Figure 40. However, in the airfoil-design problem conducted with the Robust AMF

in 5.1.4, the converged design point was slightly outside the feasible region; thus, the Robust AMF, unlike the high-fidelity model, does not find the converged design point inside the feasible region. In Section 5.2, several methods were proposed to fix this problem, but they may be somewhat temporal. Therefore, constructing a more advanced governing equation of the trust region that would not only prevent erroneous terminations of optimizations with the AMF but also lead to a converged design point within a feasible region may be the focus of future work.

APPENDIX A

EXAMPLES OF AD COMPUTATION

In this appendix, the procedures for obtaining derivatives through AD tools are presented. Here, TAPENADE is used as an AD tool. Since the explanations in the manual of TAPENADE [40] refer more to theoretical issues, this appendix focuses on practical procedures for obtaining derivatives.

Two cases are considered: computing the derivatives of an ordinary equation and computing the derivatives of a field governed by a partial differential equation (PDE). The first example is the basis for the second example, and the second example is the basis for computing the derivatives in the CFD solvers.

Finally, after a notional explanation about how to generate an adjoint code by TAPENADE, a part of the derivative code used in Section 5.3 is shown as an example for generating a derivative code of a CFD solver.

A.1 Derivatives of an Ordinary Equation

Derivatives at $(5.0, 3.0)$ of the following equation are computed using TAPENADE.

$$f = x_1 + x_2 \cos(x_1 x_2) . \tag{79}$$

Codes 1 and 2 are Fortran programs for computing f in Equation 79. As explained in 4.1.3.3, the programs should be divided into a main program part and a subroutine part that includes all subroutines that compute the function and whose primary code is referred to as a "top routine." Now, the subroutine part (i.e., Code 2, in this case) is given to TAPENADE with the setting summarized in Table 39 in order to compute

derivatives $\frac{\partial f}{\partial x_1}$, $\frac{\partial f}{\partial x_2}$ by either the FAD or RAD mode.

Table 39: Settings for computing the derivatives of Equation 79.

Mode	FAD	RAD
Dependent Output Variables	f	
Independent Input Variables	x_1, x_2	
Top Routine	func	
Differentiate Mode	Tangent	Reverse

Code 1

```
1        program main
2
3        x1=5.0
4        x2=3.0
5
6        call func(x1,x2,f)
7
8        write(6,*) "f= ", f
9
10       end
```

Code 2

```
1        subroutine func(x1,x2,f)
2
3        f = x1+x2*cos(x1*x2)
4
5        return
6        end
```

Code 4 is a subroutine that is generated by TAPENADE with the FAD mode and in which $x1d$, $x2d$, and fd are newly introduced. Code 4 has the name "func_d," and TAPENADE automatically assigns "`_d`" to the name of the top routine when it generates derivative codes with the FAD mode. In order to implement Code 4, Code 3 is newly written. In lines 6 and 7 in Code 3, the values of $x1d$ and $x2d$ should be specified. If $x1d = 1.0$ and $x2d = 0.0$, $\frac{\partial f}{\partial x_1}$ is computed as fd, and if $x1d = 0.0$ and $x2d = 1.0$, $\frac{\partial f}{\partial x_2}$ is computed as fd. The former case is shown in Code 3. Thus, in order to compute $\frac{\partial f}{\partial x_1}$ and $\frac{\partial f}{\partial x_2}$, in Code 3, subroutine Code 4 should be called twice, which is equivalent to the number of independent input variables. Note that f is also computed along with fd in Code 4.

Code 3

```
1         program main
2
3         x1=5.0
4         x2=3.0
5
6         x1d=1.0
7         x2d=0.0
8
9         call func_d(x1, x1d, x2, x2d, f, fd)
10
11        write(6,*) "f= ", f
12        write(6,*) "fd= ", fd
13
14        end
```

Code 4

```
1  C        Generated by TAPENADE     (INRIA, Tropics team)
2  C  Tapenade 2.2.2 (r1822) - 05/04/2007 13:31
3  C
4  C  Differentiation of func in forward (tangent) mode:
5  C   variations  of output variables: f
6  C   with respect to input variables: x1 x2
7        SUBROUTINE FUNC_D(x1, x1d, x2, x2d, f, fd)
8        IMPLICIT NONE
9        REAL f, fd, x1, x1d, x2, x2d
10       INTRINSIC COS
11 C
12       fd = x1d + x2d*COS(x1*x2) - x2*(x1d*x2+x1*x2d)*SIN(x1*x2)
13       f = x1 + x2*COS(x1*x2)
14 C
15       RETURN
16       END
```

Code 6 is a subroutine generated by TAPENADE using the RAD mode, in which $x1b$, $x2b$, and fb are newly introduced. Code 6 has the name "func_b," and TAPENADE automatically assigns "_b" to the name of the top routine when it generates derivative codes with the RAD mode. In order to implement Code 6, Code 5 is newly written. In line 6 of Code 5, the value of fb is specified as 1.0 in order to compute $\frac{\partial f}{\partial x_1}$ and $\frac{\partial f}{\partial x_2}$. In order to compute the derivatives of f, in Code 5, Code 6 is called once, which is the number of dependent output variables. Note that f is not computed in Code 6.

Code 5

```
1          program main
2
3          x1=5.0
4          x2=3.0
5
6          fb=1.0
7
8          call func_b(x1, x1b, x2, x2b, f, fb)
9
10 c       write(6,*) "f= ", f
11         write(6,*) "df/dx1= ", x1b
12         write(6,*) "df/dx2= ", x2b
13
14         end
```

Code 6

```
1  C        Generated by TAPENADE     (INRIA, Tropics team)
2  C  Tapenade 2.2.2 (r1822) - 05/04/2007 13:31
3  C
4  C  Differentiation of func in reverse (adjoint) mode:
5  C   gradient, with respect to input variables: f x1 x2
6  C   of linear combination of output variables: f
7        SUBROUTINE FUNC_B(x1, x1b, x2, x2b, f, fb)
8        IMPLICIT NONE
9        REAL f, fb, x1, x1b, x2, x2b
10       REAL tempb
11       INTRINSIC COS
12       tempb = -(x2*SIN(x1*x2)*fb)
13       x1b = x2*tempb + fb
14       x2b = x1*tempb + COS(x1*x2)*fb
```

```
15          fb = 0.0
16          END
```

A.2 Derivatives of a Field Governed by a Partial Differential Equation (PDE)

The derivatives of a field governed by a PDE are computed using TAPENADE, The procedure is the basis for computing the derivatives of CFD (i.e., Euler Navier-Stokes) in which a field is governed by multiple PDEs. The following two-dimensional heat-convection problem is considered:

> A square panel (199×199) is heated along the four edges and at the center. The temperatures are fixed on edges a, b, c, and d to 0.0, 20.0, 10.0, and 0.0, respectively, and 100.0 at the center. Temperatures are measured at two locations, $(50, 50)$ and $(120, 120)$, and their temperatures are referred to as "t_1" and "t_2," respectively. This problem is depicted in Figure 132.

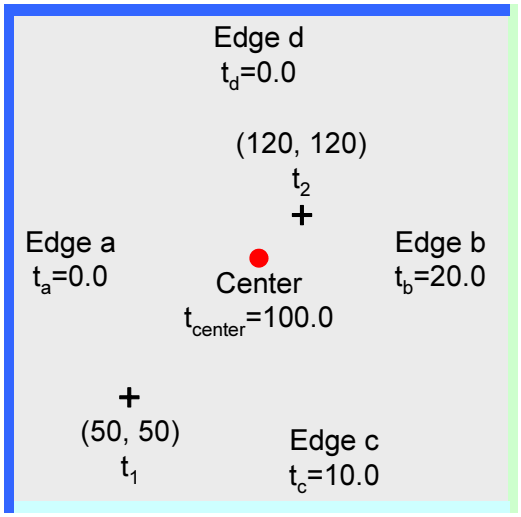


Figure 132: Two-dimensional heat-convection problem.

Then, in a stable condition, what are $\frac{\partial t_1}{\partial t_a}$, $\frac{\partial t_1}{\partial t_b}$, $\frac{\partial t_1}{\partial t_c}$, $\frac{\partial t_1}{\partial t_d}$, $\frac{\partial t_1}{\partial t_{center}}$, $\frac{\partial t_2}{\partial t_a}$, $\frac{\partial t_2}{\partial t_b}$, $\frac{\partial t_2}{\partial t_c}$, $\frac{\partial t_2}{\partial t_d}$, and $\frac{\partial t_2}{\partial t_{center}}$? Heat convection in a two-dimensional steady state is known to be governed by the following PDE:

$$\frac{\partial^2 t}{\partial x^2} + \frac{\partial^2 t}{\partial y^2} = 0. \tag{80}$$

Equation 80 is discretized by means of the finite differentiation method as follows:

$$t_{i,\,j} = \frac{1}{4}\left(t_{i+1,\,j} + t_{i-1,\,j} + t_{i,\,j+1} + t_{i,\,j-1}\right). \tag{81}$$

Codes 7 and 8 are the main and subroutine Fortran programs that solve this heat-convection problem. Since this problem can not be solved analytically, it is solved iteratively using the Gauss-Seidel method [30], starting from an arbitrary solution (lines 14-18 in Code 8). Obtaining a converged solution, i.e., the mean square residual is on the order 10^{-11}, requires a total of 25,000 iterations. The converged solution is shown in Figure 133.

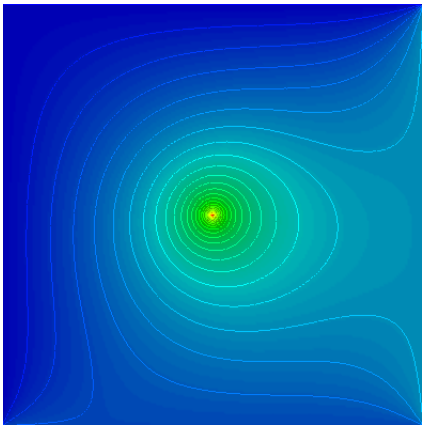


Figure 133: The converged solution of the two-dimensional heat-convection problem.

Now, the subroutine (i.e., Code 8, in this case) is given to TAPENADE with the following setting summarized in Table 40 in order to compute derivatives by either the FAD or RAD mode.

Table 40: Settings for computing the derivatives of temperature in the heat-convection problem.

Mode	FAD	RAD
Dependent Output Variables	t_1, t_2	
Independent Input Variables	t_a, t_b, t_c, t_d, t_{center}	
Top Routine	heat	
Differentiate Mode	Tangent	Reverse

Code 7

```
1         program main
2
3         implicit double precision (a-h, o-z)
4
5         common/bc/ta, tb, tc, td, tcenter
6         common/objf/t1, t2
7         common/grid/x, y, imax, jmax
8         common/temp/t
9
10        dimension x(200,200), y(200,200)
11        dimension t(200,200)
12
13  c***** Specify input variables *****
14        ta=0.0
15        tb=20.0
16        tc=10.0
17        td=0.0
18        tcenter=100.0
```

```
19
20 c***** Make grid *****
21          imax=199
22          jmax=199
23
24          do i=1,imax
25          do j=1,jmax
26          x(i,j) = 1.0/(imax-1)*(i-1)
27          y(i,j) = 1.0/(jmax-1)*(j-1)
28          end do
29          end do
30
31 c***** call subroutine *****
32          call heat
33
34 c***** output *****
35          open(1,file="grid.dat",form="unformatted")
36          write(1) imax, jmax
37          write(1) ((x(i,j), i=1, imax), j=1, jmax),
38       &          ((y(i,j), i=1, imax), j=1, jmax)
39          close(1)
40
41          open(1,file="temp.dat",form="unformatted")
42          write(1) imax, jmax, 1
43          write(1) ((t(i,j), i=1, imax), j=1, jmax)
44          close(1)
45
46          write(6,*) "t1 = ", t1
47          write(6,*) "t2 = ", t2
48
49          end
```

Code 8

```
1         subroutine heat
2
3         implicit double precision (a-h, o-z)
4
5         common/bc/ta, tb, tc, td, tcenter
6         common/objf/t1, t2
7         common/grid/x, y, imax, jmax
8         common/temp/t
9
10
11        dimension x(200,200), y(200,200)
12        dimension t0(200,200), t(200,200)
13
14  c***** initialize *****
15        do i=1,imax
16        do j=1,jmax
17        t0(i,j)=0.0
18        end do
19        end do
20
21  c***** BC at i=1 *****
22        do j=1,jmax
23        t0(1,j) = ta
24        end do
25
26  c***** BC at i=imax *****
27        do j=1,jmax
28        t0(imax,j) = tb
29        end do
30
31  c***** BC at j=1 *****
```

```
32          do i=2,imax-1
33          t0(i,1) = tc
34          end do
35
36  c***** BC at j=jmax *****
37          do i=2,imax-1
38          t0(i,jmax) = td
39          end do
40
41  c***** BC at (i,j)=(i_cent,j_cent) *****
42          i_cent = (imax+1)/2
43          j_cent = (jmax+1)/2
44          t0(i_cent,j_cent) = tcenter
45
46  c***** Solve PDE numerically *****
47
48          do i=1,imax
49          do j=1,jmax
50          t(i,j) = t0(i,j)
51          end do
52          end do
53
54          iter= 0
55          amsr = 1.0 !arbitrary number greater than 1e-12
56
57          do n=1,25000
58
59          amsr = 0.0
60
61          do i=2,imax-1
62          do j=2,jmax-1
63          t(i,j)=0.25*( t(i+1,j) + t(i-1,j) + t(i,j+1) + t(i,j-1) )
```

```
64        if (i.eq.i_cent.and.j.eq.j_cent)  t(i, j) = tcenter
65          dres = (t(i,j)-t0(i,j))**2
66          amsr  = amsr + dres
67          end do !end of j loop
68          end do !end of i loop
69
70          amsr = amsr / (imax*jmax) !mean squire residual
71          write(6,100) amsr
72
73          iter = iter + 1
74          do i=2,imax-1
75          do j=2,jmax-1
76          t0(i,j) = t(i,j)
77          end do !end of j loop
78          end do !end of i loop
79
80          end do !end of do while loop
81
82  100     format(e22.16)
83
84  c***** Compute objective functions *****
85          t1 = t(50,50)
86          t2 = t(120,120)
87
88          end
```

Code 10 is a subroutine that is generated by TAPENADE with the FAD mode and in which *tad*, *tbd*, *tcd*, *tdd*, *tcenterd*, *t*1*d*, and *t*2*d* are newly introduced. In order to implement Code 10, Code 9 is newly written. In lines 36 to 40 in Code 9, the values of *tad*, *tbd*, *tcd*, *tdd*, and *tcenterd* should be specified. If *tad* = 1.0 and the others

are zeros as in Code 9, $\frac{\partial t_1}{\partial t_a}$, $\frac{\partial t_2}{\partial t_a}$ are computed as $t1d$ and $t2d$, respectively. Thus, in order to compute all the derivatives, in Code 9, subroutine Code 10 should be called five times, which is the number of independent input variables. Note that $t1$ and $t2$ are also computed along with $t1d$ and $t2d$ in Code 10. Usually, if the FAD mode is selected, modifications in the generated derivative code (Code 10, in this case) are not required.

Code 9

```
1         program main
2
3         implicit double precision (a-h, o-z)
4
5         common/bc/ta, tb, tc, td, tcenter
6         common/objf/t1, t2
7         common/grid/x, y, imax, jmax
8         common/temp/t
9
10        dimension x(200,200), y(200,200)
11        dimension t(200,200)
12
13      COMMON /bc_d/ tad, tbd, tcd, tdd, tcenterd
14      COMMON /objf_d/ t1d, t2d
15
16
17  c***** Specify input variables *****
18        ta=0.0
19        tb=20.0
20        tc=10.0
21        td=0.0
22        tcenter=100.0
```

```
23
24 c***** Make grid *****
25          imax=199
26          jmax=199
27
28          do i=1,imax
29          do j=1,jmax
30          x(i,j) = 1.0/(imax-1)*(i-1)
31          y(i,j) = 1.0/(jmax-1)*(j-1)
32          end do
33          end do
34
35 c***** AD *****
36          tad = 1.0
37          tbd = 0.0
38          tcd = 0.0
39          tdd = 0.0
40          tcenterd = 0.0
41
42 c***** call subroutine *****
43          call heat_d
44
45 c***** output *****
46
47          write(6,*) "t1 = ", t1
48          write(6,*) "t2 = ", t2
49
50          write(6,*) "t1d = ", t1d
51          write(6,*) "t2d = ", t2d
52
53          end
```

Code 10

```
1  C        Generated by TAPENADE     (INRIA, Tropics team)
2  C  Tapenade 2.2.2 (r1822) - 05/04/2007 13:31
3  C
4  C  Differentiation of heat in forward (tangent) mode:
5  C   variations  of output variables: t1 t2
6  C   with respect to input variables: ta tb tc td tcenter
7        SUBROUTINE HEAT_D()
8        IMPLICIT NONE
9  C
10       INTEGER imax, jmax
11       DOUBLE PRECISION t(200, 200), t1, t1d, t2, t2d, ta, tad, tb, tbd,
12      +                 tc, tcd, tcenter, tcenterd, td, tdd, x(200, 200)
13      +                 , y(200, 200)
14       DOUBLE PRECISION td0(200, 200)
15       COMMON /bc/ ta, tb, tc, td, tcenter
16       COMMON /bc_d/ tad, tbd, tcd, tdd, tcenterd
17       COMMON /grid/ x, y, imax, jmax
18       COMMON /objf/ t1, t2
19       COMMON /objf_d/ t1d, t2d
20       COMMON /temp/ t
21       INTEGER i, i_cent, ii1, ii2, iter, j, j_cent, n
22       DOUBLE PRECISION amsr, dres, t0(200, 200), t0d(200, 200)
23 C
24 C
25 C
26 C
27 C
28 C***** initialize *****
29       DO i=1,imax
30         DO j=1,jmax
31           t0d(i, j) = 0.D0
```

```
32              t0(i, j) = 0.0
33            ENDDO
34          ENDDO
35          DO ii1=1,200
36            DO ii2=1,200
37              t0d(ii2, ii1) = 0.D0
38            ENDDO
39          ENDDO
40  C
41  C***** BC at i=1 *****
42          DO j=1,jmax
43            t0d(1, j) = tad
44            t0(1, j) = ta
45          ENDDO
46  C
47  C***** BC at i=imax *****
48          DO j=1,jmax
49            t0d(imax, j) = tbd
50            t0(imax, j) = tb
51          ENDDO
52  C
53  C***** BC at j=1 *****
54          DO i=2,imax-1
55            t0d(i, 1) = tcd
56            t0(i, 1) = tc
57          ENDDO
58  C
59  C***** BC at j=jmax *****
60          DO i=2,imax-1
61            t0d(i, jmax) = tdd
62            t0(i, jmax) = td
63          ENDDO
```

```
64 C
65 C***** BC at (i,j)=(i_cent,j_cent) *****
66       i_cent = (imax+1)/2
67       j_cent = (jmax+1)/2
68       t0d(i_cent, j_cent) = tcenterd
69       t0(i_cent, j_cent) = tcenter
70       DO ii1=1,200
71         DO ii2=1,200
72           td0(ii2, ii1) = 0.D0
73         ENDDO
74       ENDDO
75 C
76 C***** Solve PDE numerically *****
77 C
78       DO i=1,imax
79         DO j=1,jmax
80           td0(i, j) = t0d(i, j)
81           t(i, j) = t0(i, j)
82         ENDDO
83       ENDDO
84 C
85       iter = 0
86 Carbitrary number greater than 1e-12
87       amsr = 1.0
88 C
89       DO n=1,25000
90 C
91         amsr = 0.0
92 C
93         DO i=2,imax-1
94           DO j=2,jmax-1
95             td0(i, j) = 0.25*(td0(i+1, j)+td0(i-1, j)+td0(i, j+1)+td0(i
```

```
96        +          , j-1))
97                t(i, j) = 0.25*(t(i+1, j)+t(i-1, j)+t(i, j+1)+t(i, j-1))
98                IF (i .EQ. i_cent .AND. j .EQ. j_cent) THEN
99                  td0(i, j) = tcenterd
100                 t(i, j) = tcenter
101               END IF
102               dres = (t(i, j)-t0(i, j))**2
103               amsr = amsr + dres
104             ENDDO
105           ENDDO
106   Cend of j loop
107   Cend of i loop
108   C
109   Cmean squire residual
110           amsr = amsr/(imax*jmax)
111           WRITE(6, 100) amsr
112   C
113           iter = iter + 1
114           DO i=2,imax-1
115             DO j=2,jmax-1
116               t0(i, j) = t(i, j)
117             ENDDO
118           ENDDO
119         ENDDO
120   C
121   C***** Compute objective functions *****
122         t1d = td0(50, 50)
123         t1 = t(50, 50)
124         t2d = td0(120, 120)
125         t2 = t(120, 120)
126   Cend of j loop
127   Cend of i loop
```

```
128 C
129 Cend of do while loop
130 C
131   100   FORMAT(e22.16)
132         END
```

Code 12 is a subroutine generated by TAPENADE with the RAD mode and modified so that it does not cause any memory problem. The modified lines start with "ccc" in Code 12 (lines 29, 34, 36, 40, 52, and 53) so that they become comment lines. In this code, *tab*, *tbb*, *tcb*, *tdb*, *tcenterb*, *t1b*, and *t2b* are newly introduced. In order to implement Code 12, Code 11 is newly written. In Code 11, the converged solution is read in lines 24 to 27, and the values of *t1b* and *t2b* should be specified in lines 41 to 42. If $t1b = 1.0$ and $t2b = 0.0$ as in Code 11, $\frac{\partial t_1}{\partial t_a}$, $\frac{\partial t_1}{\partial t_b}$, $\frac{\partial t_1}{\partial t_c}$, $\frac{\partial t_1}{\partial t_d}$, and $\frac{\partial t_1}{\partial t_{center}}$ are computed as *tab*, *tbb*, *tcb*, *tdb*, and *tcenterb*, respectively. Thus, in order to compute all the derivatives, in Code 11, Code 12 should be called twice, which is the number of dependent output variables. As explained in 4.1.3.2, adjoint codes are composed of two stages: one whose direction in computing is identical to its function, and the other whose direction in computing is opposite to that of the first stage. In Code 12, the first stage starts from line 29, and the second stage starts from line 49. Some intermediate variables are stored in memory in the first stage and recalled in the second stage. In Code 12, memorizing is implemented in lines 34 and 36 using "pushinteger4" functions and recalling is implemented in line 52 using "popinteger4" functions. However, if the adjoint code is used as is (Code 12 without "ccc"), it requires a large memory size to memorize some intermediate variables, especially when computations involve an iteration process. In fact, the original derivative code (Code 12 without "ccc") can not be implemented as it is by a PC with 512 MB RAM. Therefore, instead of starting the first stage with an arbitrary solution, lines with "pushinteger4" and

"popinteger4" are commented (removed), and a converged solution is given to Code 12. (In this particular problem, since the generated adjoint code does not memorize intermediate temperatures, feeding a converged temperature is actually not required. However, since the adjoint code usually memorizes state vectors in a derivative code of a general CFD solver with the RAD, feeding a converged state vector is required. Therefore, the explanation here follows the same way as in treating a derivative code of a general CFD solver. In addition, line 54 is added in order to compensate for a function of removed line 53, but this is special for this example.) Since a set of some intermediate variables can be assumed to be identical in all iterations if a converged solution is given, one can remove both the iteration loop in the first stage and the "pushinteger4," "popinteger4" functions, as shown in Code 12. Doing so can prevent the memory problem in the RAD mode. Note that $t1$ and $t2$ are not computed along with $t1b$ and $t2b$ in Code 12.

Code 11

```
1         program main
2
3         implicit double precision (a-h, o-z)
4
5         common/bc/ta, tb, tc, ts, tcenter
6         common/objf/t1, t2
7         common/grid/x, y, imax, jmax
8         common/temp/t
9
10        dimension x(200,200), y(200,200)
11        dimension t(200,200)
12
13      common/bc_b/ tab, tbb, tcb, tdb, tcenterb
14      common/objf_b/ t1b, t2b
```

```
15
16 c***** Specify input variables *****
17          ta=0.0
18          tb=20.0
19          tc=10.0
20          td=0.0
21          tcenter=100.0
22
23 c***** Read converged solution *****
24          open(1,file="temp.dat", form="unformatted")
25          read(1) imax, jmax
26          read(1) ((t(i,j), i=1, imax), j=1, jmax)
27          close(1)
28
29 c***** Make grid *****
30          imax=199
31          jmax=199
32
33          do i=1,imax
34          do j=1,jmax
35          x(i,j) = 1.0/(imax-1)*(i-1)
36          y(i,j) = 1.0/(jmax-1)*(j-1)
37          end do
38          end do
39
40 c***** AD *****
41          t1b=1.0
42          t2b=0.0
43
44 c***** call subroutine *****
45          call heat_b
46
```

```
47 c***** output *****
48
49 C        write(6,*) "t1 = ", t1
50 C        write(6,*) "t2 = ", t2
51
52         write(6,*) "tab = ", tab
53         write(6,*) "tbb = ", tbb
54         write(6,*) "tcb = ", tcb
55         write(6,*) "tdb = ", tdb
56         write(6,*) "tcenterb = ", tcenterb
57
58         end
```

Code 12

```
1  C        Generated by TAPENADE     (INRIA, Tropics team)
2  C  Tapenade 2.2.2 (r1822) - 05/04/2007 13:31
3  C
4  C  Differentiation of heat in reverse (adjoint) mode:
5  C   gradient, with respect to input variables: ta tb tc td tcenter
6  C                t1 t2
7  C   of linear combination of output variables: t1 t2
8        SUBROUTINE HEAT_B()
9        IMPLICIT NONE
10 C
11       INTEGER imax, jmax
12       DOUBLE PRECISION tb0(200, 200)
13       DOUBLE PRECISION t(200, 200), t1, t1b, t2, t2b, ta, tab, tb, tbb,
14      +                 tc, tcb, tcenter, tcenterb, td, tdb, x(200, 200)
15      +                 , y(200, 200)
16       COMMON /bc/ ta, tb, tc, td, tcenter
17       COMMON /bc_b/ tab, tbb, tcb, tdb, tcenterb
18       COMMON /grid/ x, y, imax, jmax
```

```
19        COMMON /objf/ t1, t2
20        COMMON /objf_b/ t1b, t2b
21        COMMON /temp/ t
22        INTEGER branch, i, i_cent, ii1, ii2, iter, j, j_cent, n
23        DOUBLE PRECISION amsr, dres, t0(200, 200), t0b(200, 200), tempb
24  C
25  C***** BC at (i,j)=(i_cent,j_cent) *****
26        i_cent = (imax+1)/2
27        j_cent = (jmax+1)/2
28  C
29  ccc       DO n=1,25000
30  C
31          DO i=2,imax-1
32            DO j=2,jmax-1
33              IF (i .EQ. i_cent .AND. j .EQ. j_cent) THEN
34  ccc               CALL PUSHINTEGER4(2)
35              ELSE
36  ccc               CALL PUSHINTEGER4(1)
37              END IF
38            ENDDO
39          ENDDO
40  ccc       ENDDO
41        DO ii1=1,200
42          DO ii2=1,200
43            tb0(ii2, ii1) = 0.D0
44          ENDDO
45        ENDDO
46        tb0(120, 120) = t2b
47        tb0(50, 50) = tb0(50, 50) + t1b
48        tcenterb = 0.D0
49        DO n=25000,1,-1
50          DO i=imax-1,2,-1
```

```
51            DO j=jmax-1,2,-1
52  ccc             CALL POPINTEGER4(branch)
53  ccc             IF (.NOT.branch .LT. 2) THEN
54              IF (i .EQ. i_cent .AND. j .EQ. j_cent) THEN
55                tcenterb = tcenterb + tb0(i, j)
56                tb0(i, j) = 0.D0
57              END IF
58              tempb = 0.25*tb0(i, j)
59              tb0(i+1, j) = tb0(i+1, j) + tempb
60              tb0(i-1, j) = tb0(i-1, j) + tempb
61              tb0(i, j+1) = tb0(i, j+1) + tempb
62              tb0(i, j-1) = tb0(i, j-1) + tempb
63              tb0(i, j) = 0.D0
64            ENDDO
65          ENDDO
66        ENDDO
67        DO ii1=1,200
68          DO ii2=1,200
69            t0b(ii2, ii1) = 0.D0
70          ENDDO
71        ENDDO
72        DO i=imax,1,-1
73          DO j=jmax,1,-1
74            t0b(i, j) = t0b(i, j) + tb0(i, j)
75            tb0(i, j) = 0.D0
76          ENDDO
77        ENDDO
78        tcenterb = tcenterb + t0b(i_cent, j_cent)
79        t0b(i_cent, j_cent) = 0.D0
80        tdb = 0.D0
81        DO i=imax-1,2,-1
82          tdb = tdb + t0b(i, jmax)
```

```
 83            t0b(i, jmax) = 0.D0
 84          ENDDO
 85          tcb = 0.D0
 86          DO i=imax-1,2,-1
 87            tcb = tcb + t0b(i, 1)
 88            t0b(i, 1) = 0.D0
 89          ENDDO
 90          tbb = 0.D0
 91          DO j=jmax,1,-1
 92            tbb = tbb + t0b(imax, j)
 93            t0b(imax, j) = 0.D0
 94          ENDDO
 95          tab = 0.D0
 96          DO j=jmax,1,-1
 97            tab = tab + t0b(1, j)
 98            t0b(1, j) = 0.D0
 99          ENDDO
100          t1b = 0.D0
101          t2b = 0.D0
102   Cend of j loop
103   Cend of i loop
104   C
105   Cend of do while loop
106   C
107    100   FORMAT(e22.16)
108          END
```

Finally, derivatives computed by the FD, the FAD, and the RAD are summarized in Table 41. Since the results from each method are almost equivalent, they are printed in the same columns. However, the computation times differ.

Table 41: Derivatives in the heat-convection problem.

	FD ($\Delta t = 1.0$)	FAD	RAD
Time Required [sec]	671	410	151
$\frac{\partial t_1}{\partial t_a}$	0.4161456		
$\frac{\partial t_1}{\partial t_b}$	0.0493895		
$\frac{\partial t_1}{\partial t_c}$	0.4161456		
$\frac{\partial t_1}{\partial t_d}$	0.0493895		
$\frac{\partial t_1}{\partial t_{center}}$	0.0685601		
$\frac{\partial t_2}{\partial t_a}$	0.1146653		
$\frac{\partial t_2}{\partial t_b}$	0.2794076		
$\frac{\partial t_2}{\partial t_c}$	0.1146653		
$\frac{\partial t_2}{\partial t_d}$	0.2794076		
$\frac{\partial t_2}{\partial t_{center}}$	0.2112883		

A.3 Derivative Code of a CFD Solver

If a CFD flow solver whose structure is shown in Figure 134 is differentiated by TAPENADE with the RAD mode, an adjoint code, shown in Figure 135, is yield. Here, aerodynamic coefficients such as C_L and C_D are specified as output variables, and grid locations x, y, and z, as input variables in TAPENADE. Note that lines calling "POPREAL8()" and "POPREAL8()" always appear as pairs in the adjoint code, shown in Figure 135.

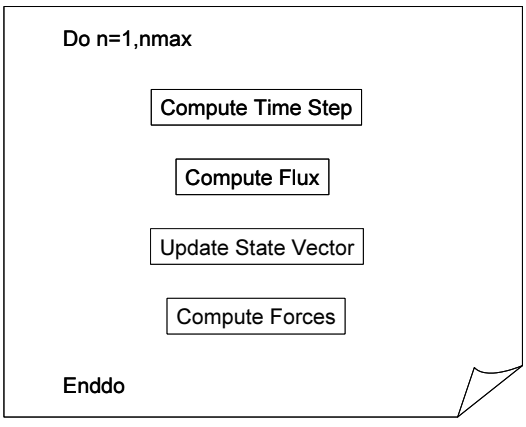


Figure 134: The CFD flow solver.

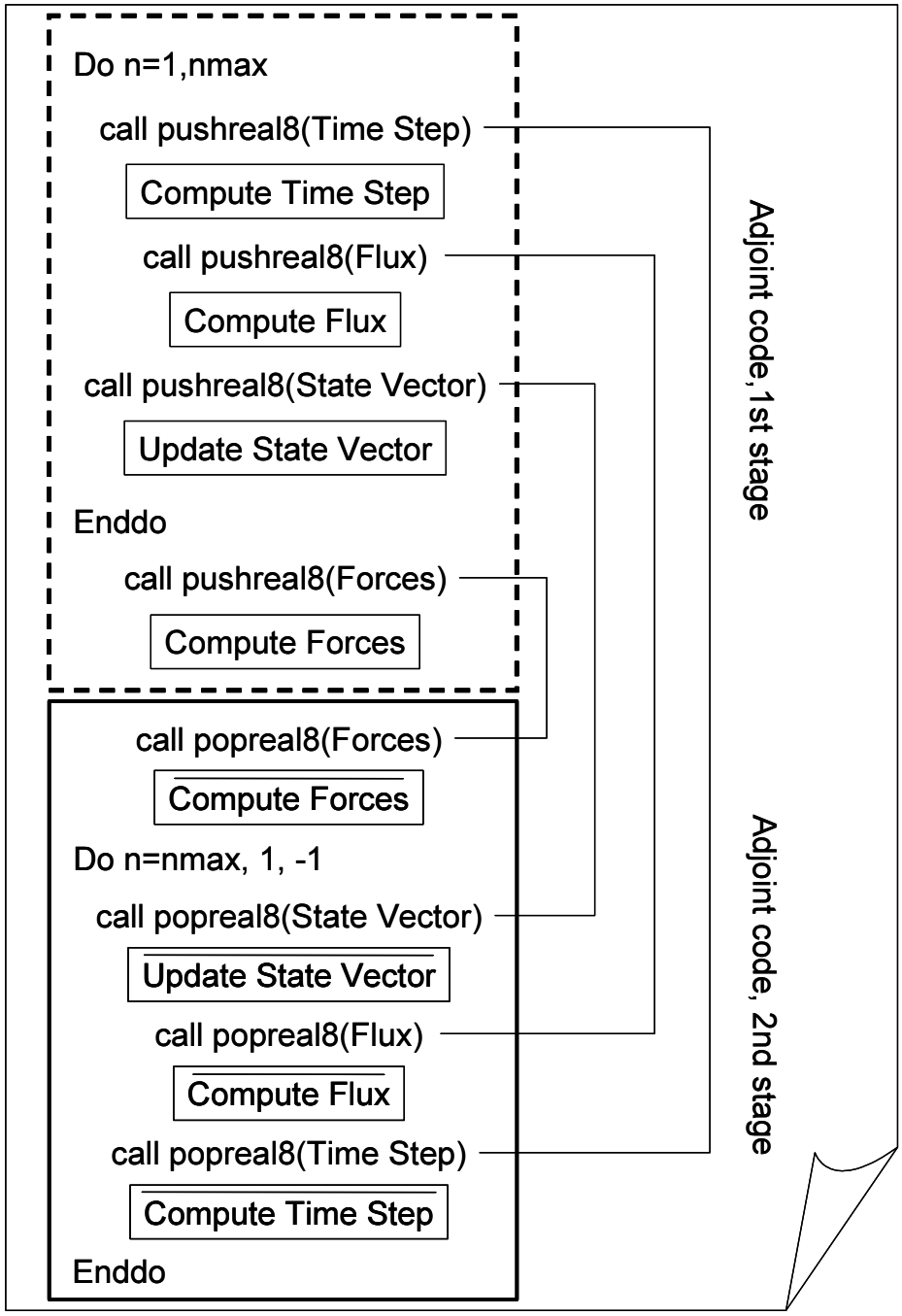


Figure 135: The adjoint code generated by TAPENADE before modifications.

In order to prevent the memory problem, the adjoint code shown in 135 should be modified as in an adjoint code shown in Figure 136.

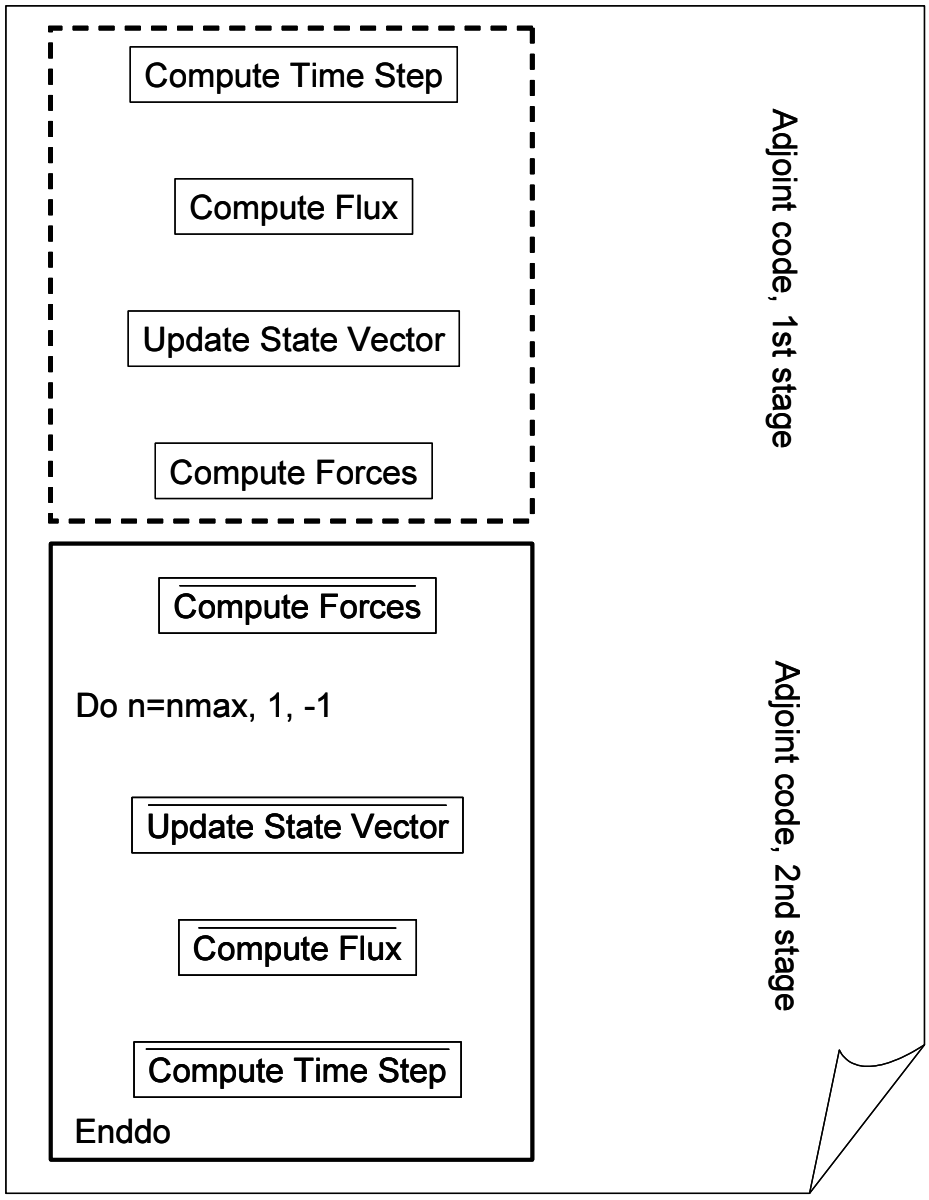


Figure 136: The adjoint code generated by TAPENADE after modifications.

This procedure above was implemented in this study. The top routine part of the derivative code used in Section 5.3 is shown as in Code 13, which is generated by TAPENADE with the RAD mode and modified so that it prevents the memory problem. The original CFD solver given to TAPENADE has a structure shown in Figure 137, in which subroutines under "third routines" are omitted.

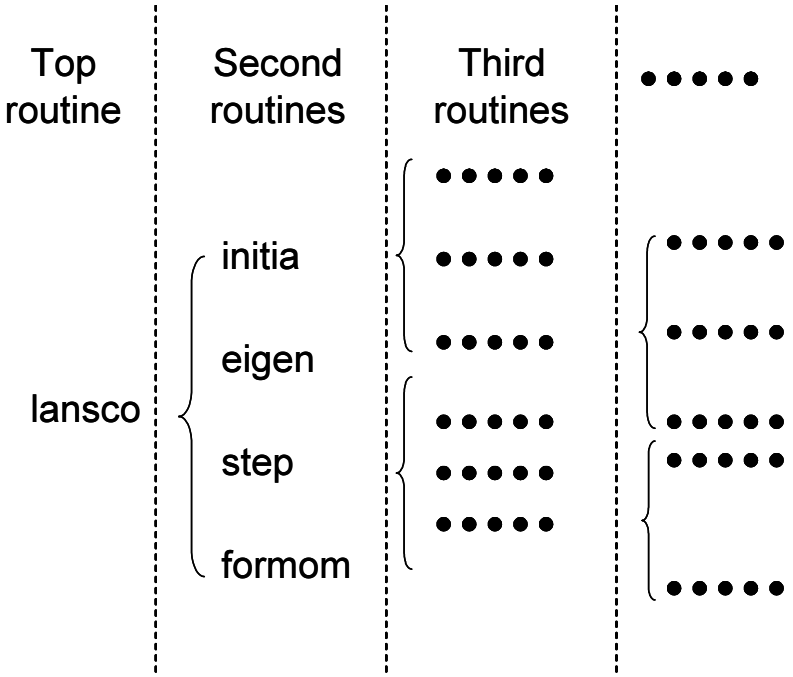


Figure 137: The source code structure of LANS.

Lines starting from "ccc" in Code 13 (lines 53, 54, 56, 66, 68, 69, and 70 for removing "PUSH" functions, 99, 100, 101, 103, 105, 107, and 108 for removing "POP" functions) are those that are removed after the modification. The first stage of the adjoint code starts from line 65, and the second from line 98. Since some intermediate variables computed in the first stage of the adjoint code should be required in the second stage in this case, a converged state vector should be given to the adjoint code, as pointed out in Section 4.2. Note that the top routine part of the derivative code are only modified in this study, but modifying other parts of the derivative code may be possible. If so, one can expect more savings in CPU time and memory size for implementing the derivative code.

Code 13

```
1 C          Generated by TAPENADE     (INRIA, Tropics team)
2 C  Tapenade 2.2.2 (r1822) -- 05/04/2007 13:31
```

```
3  C
4  C  Differentiation of lansco in reverse (adjoint) mode:
5  C   gradient, with respect to input variables: x y z cl cd
6  C   of linear combination of output variables: cl cd
7        SUBROUTINE LANSCO_B()
8        IMPLICIT NONE
9        INTEGER jdim, kdim, ldim, mdim, nd, nv
10       PARAMETER (jdim=300, kdim=100, ldim=100, mdim=300, nd=6, nv=5)
11       INTEGER ilhs, invisc, iread, irhs, iroe, iwrit, jm, jmax, jstep,
12      +         jtel, jteu, ke1, km, kmax, lamin, lm, lmax, nc, nc1, ngri
13      +         , nmax, np
14       REAL alp, cd, cdb, cl, clb, cnbr, dt, dtb, dx1, dy1, dz1, fd(nv),
15      +     fsmach, fv(nv), gami, gamma, gd, hd, hdb, hdx, hdxb, hdy,
16      +     hdyb, hdz, hdzb, p(mdim, mdim, nv), pb(mdim, mdim, nv), pi,
17      +     pr, q(jdim, kdim, ldim, nd), qb(jdim, kdim, ldim, nd), re, rk
18      +     , rm, rmue, s(jdim, kdim, ldim, nv), sb(jdim, kdim, ldim, nv)
19      +     , smr, smu, x(jdim, kdim, ldim), xb(jdim, kdim, ldim), xt, xy
20      +     (jdim, kdim, ldim, 3, 3), xyb(jdim, kdim, ldim, 3, 3), y(jdim
21      +     , kdim, ldim), yb(jdim, kdim, ldim), yt, z(jdim, kdim, ldim)
22      +     , zb(jdim, kdim, ldim), zt
23       COMMON /aero/ cl, cd
24       COMMON /aero_b/ clb, cdb
25       COMMON /base/ nmax, jmax, kmax, lmax, jm, km, lm, dt, gamma, gami
26      +, smu, fsmach, dx1, dy1, dz1, fv, fd, hd, alp, gd, hdx, hdy, hdz,
27      +rm, cnbr, pi, invisc, lamin, np, jstep, smr, ilhs, irhs, iroe
28       COMMON /base_b/ dtb, hdb, hdxb, hdyb, hdzb
29       COMMON /count/ nc, nc1
30       COMMON /edge/ jteu, jtel, ke1
31       COMMON /read/ iread, iwrit, ngri
32       COMMON /var0/ s
33       COMMON /var0_b/ sb
34       COMMON /var1/ x, y, z
```

```
35        COMMON /var1_b/ xb, yb, zb
36        COMMON /var2/ p
37        COMMON /var2_b/ pb
38        COMMON /var3/ xt, yt, zt, xy
39        COMMON /var3_b/ xyb
40        COMMON /vars/ q
41        COMMON /vars_b/ qb
42        COMMON /vis/ re, pr, rmue, rk
43        INTEGER ii1, ii2, ii3, ii4, kh, kh2, krdum, krs(5), n, ndum
44        REAL cmp, cmr, cmy, cx, cy, cz, sxx(jdim, kdim), sxy(jdim, kdim),
45       +     sxz(jdim, kdim), syy(jdim, kdim), syz(jdim, kdim), szz(jdim,
46       +     kdim), x0, y0, z0
47  C
48  C
49  C
50  C*********************  initialization  ******************************
51  C
52        nc1 = 0
53  ccc       CALL PUSHREAL4ARRAY(xy, jdim*kdim*ldim*3**2)
54  ccc       CALL PUSHREAL4ARRAY(q, jdim*kdim*ldim*nd)
55        CALL INITIA()
56  ccc       CALL PUSHREAL4(cnbr)
57  C
58  C***********  compute maximum eigenvalue and courant number  ***********
59  C
60        CALL EIGEN()
61  C
62  C*********************  implicit integration  ************************
63  C
64  C
65  ccc       DO n=1,nmax
66  ccc         CALL PUSHINTEGER4(nc)
```

```
67            nc = n + nc1
68  ccc           CALL PUSHREAL4ARRAY(s, jdim*kdim*ldim*nv)
69  ccc           CALL PUSHREAL4ARRAY(p, mdim**2*nv)
70  ccc           CALL PUSHREAL4ARRAY(q, jdim*kdim*ldim*nd)
71            CALL STEP()
72  C
73  ccc        ENDDO
74          CALL FORMOM_B(jtel, jteu, 2, kmax, 1, invisc, x0, y0, z0, cx, cy,
75         +                  cz, cl, clb, cd, cdb, cmr, cmp, cmy, re, fsmach, 1.
76         +                  , alp, gamma, sxx, syy, szz, sxy, sxz, syz, 1)
77          dtb = 0.0
78          hdb = 0.0
79          hdxb = 0.0
80          hdyb = 0.0
81          hdzb = 0.0
82          DO ii1=1,nv
83            DO ii2=1,mdim
84              DO ii3=1,mdim
85                pb(ii3, ii2, ii1) = 0.0
86              ENDDO
87            ENDDO
88          ENDDO
89          DO ii1=1,nv
90            DO ii2=1,ldim
91              DO ii3=1,kdim
92                DO ii4=1,jdim
93                  sb(ii4, ii3, ii2, ii1) = 0.0
94                ENDDO
95              ENDDO
96            ENDDO
97          ENDDO
98          DO n=nmax,1,-1
```

```
 99 ccc           CALL POPREAL4ARRAY(q, jdim*kdim*ldim*nd)
100 ccc           CALL POPREAL4ARRAY(p, mdim**2*nv)
101 ccc           CALL POPREAL4ARRAY(s, jdim*kdim*ldim*nv)
102             CALL STEP_B()
103 ccc           CALL POPINTEGER4(nc)
104           ENDDO
105 ccc         CALL POPREAL4(cnbr)
106           CALL EIGEN_B()
107 ccc         CALL POPREAL4ARRAY(q, jdim*kdim*ldim*nd)
108 ccc         CALL POPREAL4ARRAY(xy, jdim*kdim*ldim*3**2)
109           CALL INITIA_B()
110           clb = 0.0
111           cdb = 0.0
112           END
```

REFERENCES

[1] "Intelligent Light FIELDVIEW." `http://www.ilight.com/`; Accessed March 2008.

[2] *MATLAB Optimization Toolbox 4 User's Guide*.

[3] "TOP500 Supercomputing Sites." `http://www.top500.org/`; Accessed March 2008.

[4] *Defense Acquisition Guidebook*. Management Concepts, 2006.

[5] Abbott, H. I., *Theory of Wing Sections*. Dover, 1959.

[6] Alexandrov, N. M., "Robustness properties of a trust region framework for managing approximations in engineering optimization," in *6th AIAA/-NASA/USAF Multidisciplinary Analysis and Optimization Symposium*, AIAA Paper No. 96-4102, 1996.

[7] Alexandrov, N. M., "Optimization with variable-fidelity models applied to wing design," in *38th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper No. 2000-0841, 2000.

[8] Alexandrov, N. M., "An evolutionary risk adjusting model fusion framework for optimizing models with variable fidelity," in *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA Paper No. 2004-4577, 2004.

[9] Anderson, D. J., *Modern Compressible Flow*. McGraw-Hill Publish Company, 1990.

[10] Anderson, D. J., *Aircraft Performance and Design*. McGraw-Hill, 1999.

[11] Anderson, D. J., *Fundamentals of Aerodynamics Third Edition*, pp. 597–602. McGraw-Hill Publish Company, 2001.

[12] Argonne National Laboratory, "ADIC Resource Center." `http://www-new.mcs.anl.gov/adic/`; Accessed October 2007.

[13] Azen, S. P., *Successive approximation by quadratic fitting as applied to optimization problems*. Rand corp santa monica calif, 1966.

[14] Batina, T. J., "Unsteady euler airfoil solutions using unstructured dynamic meshes," in *AIAA 27th Aerospace Sciences Meeting*, AIAA Paper No. 89-0115, 1989.

[15] Bertin, J. J., *Aerodynamics for Engineers*, pp. 256–259. Pretice Hall, 2002.

[16] Bertin, J. J., *Aerodynamics for Engineers*, pp. 260–278. Pretice Hall, 2002.

[17] Blom, J. F., "Considerations on the spring analogy," *International journal for numerical methods in fluids*, vol. 32, pp. 647–668, 2000.

[18] Broyden, C. G., "The convergence of a class of double-rank minimization algorithms," *IMA Journal of Applied Mathematics*, vol. 6, pp. 76–90, 1970.

[19] Chang, K. J., "Sensitivity-based scaling for approximating structural response," *Journal of Aircraft*, vol. 30, no. 2, pp. 283–288, 1993.

[20] Chung, H., "Using gradients to construct cokriging approximation models for high-dimensional design optimization problems," in *AIAA 40th Aerospace Sciences Meeting*, 2002.

[21] Conn, A. R., "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM Journal of Numerical Analysis*, vol. 25, no. 2, pp. 433–464, 1988.

[22] Conn, A. R., "A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal*, vol. 28, no. 2, pp. 545–572, 1991.

[23] Doxygen, "ADMC++: Automatic Differentiation of Matlab and C++." `http://admcpp.sourceforge.net/index.html`; Accessed October 2007.

[24] Dudley, J., "Multidisciplinary optimization of the high-speed civil transport," in *AIAA 33rd Aerospace Sciences Meeting*, AIAA Paper No. 95-0124, 1995.

[25] Ekman, M., "An in-depth look at computer performance growth," tech. rep., Chalmers University of Technology, 2004.

[26] El-Beltagy, A. M., "Multilevel and multiobjective optimization in multidisciplinary design," in *6th AlAA/NASA/lSSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA Paper No. 1996-4122, 1996.

[27] Eriksson, L. E., "Generation of boundary grids around wing-body configurations using transfinite interpolation," *AIAA Journal*, vol. 20, pp. 1313–1320, 1982.

[28] FENIX, "benchmark.txt." `http://www.fenix.ne.jp/~G-HAL/comp/benchmark.txt`; Accessed January 2008.

[29] Ferziger, H. J., *Numerical Methods for Engineering Application.* Wiley-Interscience, 1998.

[30] Ferziger, H. J., *Numerical Methods for Engineering Application.* Wiley-Interscience, 1998.

[31] Fletcher, R., "A new approach to variable metric algorithms," *Computer Journal*, vol. 13, pp. 317–322, 1970.

[32] Forrest, W. B. I., *Implementing six sigma*. Wiley-Interscience, 1999.

[33] Forrest, W. B. I., *Implementing six sigma*. Wiley-Interscience, 1999.

[34] Fujii, K., "Navier-stokes simulations of transonic flows over a practical wing configuration," *AIAA Journal*, vol. 25, pp. 369–370, 1987.

[35] Fujii, K., *Numerical Methods for Computational Fluid Dynamics*. University of Tokyo Press, 1994.

[36] Gano, E. S., *Simulation-Based Design Using Variable Fidelity Optimization*. PhD thesis, Univeristy of Notre Dame, 2005.

[37] Giunta, A. A., "Implementation of a trust region model management strategy in the dakota optimization toolkit," in *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA Paper No. 2000-4935, 2000.

[38] Goldfarb, D., "A family of variable metric updates derived by variational means," *Mathematics of Computing*, vol. 24, pp. 23–26, 1970.

[39] Haftka, R. T., *Optimization*. Kluwer, 1993.

[40] Hascoet, L., *TAPENADE2.1 user's guide*, 2004.

[41] Haykin, S., *Neural Networks A Comprehensive Foundation*. McGraw-Hill Publish Company, 2003.

[42] Hecht, F., "BAMG (english)." `http://www-c.inria.fr/gamma/cdrom/www/bamg/eng.htm`; Accessed October 2007.

[43] Hicks, M. R., "Wing design by numerical optimization," AIAA Paper No. 77-1247, 1977.

[44] Hopkins, J. E., "Aerodynamic characteristics of several cranked leading-edge wing-body combinations at mach numbers from 0.4 to 2.94," tech. rep., NASA, Ames research center, 1967.

[45] Hutchison, M. G., "Aerodynamic optimization of and hsct configuration using variable-complexity modeling," in *AIAA 31th Aerospace Sciences Meeting*, AIAA Paper No. 93-0101, 1993.

[46] INRIA, "TAPENADE On-line Automatic Differentiation Engine." `http://tapenade.inria.fr:8080/tapenade/index.jsp`; Accessed October 2007.

[47] Jameson, A., "Iterative solution of transonic flows over airfoils and wings, including flows at mach 1," *Communications in Pure and Applied Mathematics*, vol. 27, pp. 283–309, 1974.

[48] Jameson, A., "Aerodynamic design via control theory," *Journal of Scientific Computing*, vol. 3, no. 3, pp. 233–260, 1988.

[49] Jones, W. P., "Theprediction of laminarization with a two-equation model of turbulence," *International Journal of Heat Mass Transfer*, vol. 15, pp. 301–314, 1972.

[50] Kim, H., "Aerodynamic optimization of supersonic transport wing using unstructured adjoint method," *AIAA Journal*, vol. 39, pp. 1011–1020, 2001.

[51] Kirby, M., *A Methodology for Technology, Identification, Evaluation, and Selection in Conceptual and Preliminary Aircraft Design.* PhD thesis, Georgia Institute of Technology, 2001.

[52] Li, C. P., "Numerical solution of viscous reacting blunt body flows of a multicomponent mixture," AIAA Paper No. 73-202, 1973.

[53] Lo, S. H., "A new mesh generation scheme for arbitrary planar domains," *International Journal for Numerical Methods in Engineering*, vol. 21, pp. 1403–1426, 1985.

[54] Marduel, X., "Optimization using variable fidelity solvers: Exploration of an approximation management framework for aerodynamic shape optimization.," in *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA Paper No. 2002-5595, 2002.

[55] Marduel, X., "Variable-fidelity optimization: Efficiency and robustness," *Optimization and Engineering*, vol. 7, no. 4, pp. 477–500, 2006.

[56] Markou, A. G., "The ortho-semi-torsional(ost) spring analogy method for 3d mesh moving boundary problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 196, pp. 747–765, 2007.

[57] Mavris, D. N., "Methodology for examining the simultaneous impact of requirements, vehicle characteristics, and technologies on military aircraft design," in *22 nd Congress of the International Council on the Aeronautical Science*, 2000.

[58] Mohammadi, B., "Sukhoi S-21 Supersonic Business Jet." `http://www.aeronautics.ru/s21.htm`; Accessed February 2008.

[59] Murman, E., "Calculation of plane steady transonic flows," *AIAA Journal*, vol. 9, pp. 114–121, 1971.

[60] Naumann, U., "OpenAD Summary." `http://www-unix.mcs.anl.gov/OpenAD/summary.html`; Accessed October 2007.

[61] Nobre, M. M., "Application of bayesian kriging to subsurface characterization," *Canadian geotechnical journal*, vol. 29, pp. 589–598, 1992.

[62] Obayashi, S., "An approximate lu factorization method for the compressible navier-stokes equation," *Journal of Computational Physics*, vol. 63, pp. 157–167, 1986.

[63] Peraire, J., "Adaptive remeshing for compressible flow computations," *Journal of Computational Physics*, vol. 72, pp. 449–466, 1987.

[64] Piegl, L., *The NURBS Book*. Springer, 1997.

[65] Rasmussen, E. C., "The gaussian processes web site." `http://www.gaussianprocess.org/`; Accessed January 2008.

[66] Raymer, P. D., *Aircraft Design: A Conceptual Approach Third Edition*. AIAA Education Series, 1999.

[67] Reuther, A., "Aerodynamic shape optimization of complex aircraft via an adjoint formulation," in *AIAA 34th Aerospace Sciences Meeting*, AIAA Paper No. 96-0094, 1996.

[68] Rodriguez, J. F., "Convergence of trust region augmented lagradian methods using variable fidelity approximation data," *Structual Optimization*, vol. 15, pp. 1–7, 1998.

[69] Roe, P. L., "Approximate riemann solvers, parameter vectors, and difference scheme," *Journal of Computational Physics*, vol. 43, pp. 357–372, 1981.

[70] Roskam, J., *Airplane Design Parts I through VIII*. Roskam Aviation and Engineering Corporation, 1985.

[71] Saaris, R. G., *A502I User's Guide-PAN AIR Technology Program for Solving Potential Flow about Arbitrary Configurations*. Boeing, 1992.

[72] Sacks, J., "Design and analysis of computer experiments," *Statistical Science*, vol. 4, pp. 409–423, 1989.

[73] Sankar, N. L., "Ae 3903/4903 airfoil design." `http://www.ae.gatech.edu/people/lsankar/AE6022/fpe.for`; Accessed October 2007.

[74] Schittkowski, K., *Lecture Notes in Economics and Mathematical Systems*. Springer, 1987.

[75] Shanno, D. F., "Conditioning of quasi-newtonmethods for function minimization," *Mathematics of Computing*, vol. 24, pp. 647–656, 1970.

[76] Spragle, G. S., "Computation of 2d unstructured grid generation techniques," AIAA Paper No. 91-0726, 1991.

[77] Tannehill, C. J., *Computational Fluid Mechanics and Heat Transfer*. Taylor and Francis, 1997.

[78] The Mathematics and Computer Science Division at Argonne National Laboratory and the Center for Research on Parallel Computation at Rice University, "ADIFOR: Automatic Differentiation of Fortran Codes." `http://www-unix.mcs.anl.gov/autodiff/ADIFOR/`; Accessed October 2007.

[79] van Leer, B., "Toward the ultimate conservative difference scheme," *Journal of Computational Physics*, vol. 23, pp. 276–299, 1977.

[80] Vanderplaats, N. G., *Numerical Optimization Techniques for Engineering Design, Third Eddition*, pp. 250–254. VR and D, 2001.

[81] Vanderplaats, N. G., *Numerical Optimization Techniques for Engineering Design, Third Eddition*, pp. 130–137. VR and D, 2001.

[82] Venik's Aviation, "NSC2KE (english)." `http://www-c.inria.fr/gamma/cdrom/www/nsc2ke/eng.htm`; Accessed October 2007.

[83] Viviand, H., "Test cases for inviscid flow field methods," tech. rep., AGARD, 1985.

[84] Wengert, R. E., "A simple automatic derivative evaluation program," *Communications of the ACM*, vol. 7, no. 8, pp. 463–464, 1964.

[85] Whitcomb, T. R., "An airfoil shape for efficient flight at supercritical mach numbers," tech. rep., NASA, 1965.

[86] Wilcox, D. C., *Turbulence Modeling for CFD*. DCW Industries, 1993.

VITA

Takemiya Tetsushi was born in Kyoto, Japan in 1975. He attended Waseda University and graduated with his Bachelor of Science in Mechanical Engineering in 1999. He joined the University of Tokyo and graduated with his Master of Science in Aerospace Engineering in 2001.

In 2002, he joined the Aerospace Systems Design Laboratory at the Georgia Institute of Technology to pursue his doctorate.

He loves music and drinking.